Approximately Optimum Search Trees in External Memory Models

by

Oliver Grant

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We examine optimal and near optimal solutions to the classic binary search tree problem of Knuth. We are given a set of n keys (originally known as words), $B_1, B_2, ..., B_n$ and $2n+1$ frequencies. $p_1, p_2, ..., p_n$ represent the probabilities of searching for each given key, and $q_0, q_1, ..., q_n$ represent the probabilities of searching in the gaps between and outside of these keys. We have that $\sum_{i=0}^{n} q_i + \sum_{i=1}^{n} p_i = 1$. We also assume without loss of generality that $q_{i-1} + p_i + q_i \neq 0$ for any $i \in \{1, ..., n\}$. The keys must make up the internal nodes of the tree while the gaps make up the leaves. Our goal is to construct a binary search tree such that expected cost of search is minimized. First, we re-examine an approximate solution of Güttler, Mehlhorn and Schneider which was shown to have a worst case bound of $c \cdot H + 2$ where $c \geq \frac{1}{H(\frac{1}{3}, \frac{2}{3})} \approx 1.08$, and $H = \sum_{i=1}^{n} p_i \cdot \lg(\frac{1}{p_i}) + \sum_{j=0}^{n} q_i \cdot \lg(\frac{1}{q_j})$ is the entropy of the distribution. We give an improved worst case bound on the heuristic of $H + 4$. Next, we examine the optimum binary search tree problem under a model of external memory. We use the Hierarchical Memory Model of Aggarwal et al. The model has an unlimited number of registers, $R_1, R_2, ...$ each with its own location in memory (a positive integer). We have a set of memory sizes $m_1, m_2, ..., m_l$ which are monotonically increasing. Each memory level has a finite size except m_l which we assume has infinite size. Each memory level has an associated cost of access $c_1, c_2, ..., c_l$. We assume that $c_1 < c_2 < ... < c_l$. We propose two approximate solutions which run in $O(n)$ time where n is the number of words in our data set. Using these methods, we improve upon a bound given in Thite's 2001 thesis under the related HMM_2 model in the approximate setting. We also examine the related problem of binary trees on multisets of probabilities where keys are unordered and we do not differentiate between which probabilities must be leaves, and which must be internal nodes. We provide a simple $O(n \lg(n))$ algorithm that is within an additive $\frac{n+1}{2n}$ of optimal on a multiset of n keys.

Acknowledgements

First and foremost, I would like to thank my supervisor Ian Munro for guiding me during my graduate studies here at the University of Waterloo. Learning from him was a great privilege. I will fondly remember our talks about data structures, algorithms, and life in general.

I would also like to thank Mordecai Golin. Conversations with Mordecai significantly helped in the simplification of the proof in 5.2.1 and helped shape Chapter 3.

I would also like to thank my lab mate Alexandre Daigle for thoroughly proof reading my first thesis draft, and spending countless hours working on whiteboard proofs with me. Also thanks to colleagues Hisham El-Zein, Dimitrios Skrepetos and Simon Pratt for working on problems with me at various times throughout my studies.

Finally, thanks to my committee members Anna Lubiw and Eric Blais for their constructive criticism of my work.

Dedication

This is dedicated to my partner Melissa.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we provide an introduction to binary search trees and the optimum binary search tree problem. We also give motivation for studying binary search trees and give an overview of the work presented in this thesis. We note that for the entirety of this work, we will use lg to represent log_2.

1.1 Binary Search Trees

A binary search tree is a simple structure used to store key-value pairs. It was invented in the late 1950s and early 1960s and is generally attributed to the combined efforts of Windley [58], Booth and Colin [14] and Hibbard [34]. In general, a binary search tree (BST) allows for quick binary searches through data for a specific key. There is a total ordering over the keys of the tree which are typically numbers or words. The value of a node in the BST usually represents some piece of important information, and is often a pointer to a large structure somewhere else in memory. Each BST node has at most two children which are generally labelled as the *left* and *right* children. All nodes in the subtree of the *left* child of a specific node p have a key strictly less than the key of p. Similarly, nodes in the subtree of the *right* child of p have a key strictly greater than the key of p. A pointer is typically stored to the root node. Search begins from this root node and is done by recursively searching in either the *left* or *right* child of a node; stopping if the node being searched has the correct key, or if the node reached has no children. When searching, we typically make a single comparison at each node visited.

1.2 The Optimum Binary Search Tree Problem

Knuth first proposed the optimum binary search tree problem in 1971 [43]. We are given a set of n keys (originally known as words), $B_1, B_2, ..., B_n$ and $2n+1$ frequencies. $p_1, p_2, ..., p_n$ represent the probabilities of searching for each given key, and $q_0, q_1, ..., q_n$ represent the probabilities of searching in the gaps between and outside of these keys. We have that

$$\sum_{i=0}^{n} q_i + \sum_{i=1}^{n} p_i = 1$$

We also assume without loss of generality that $q_{i-1} + p_i + q_i \neq 0$ for any $i \in \{1, ..., n\}$. Otherwise, we could simply solve the problem with key p_i removed. The keys must make up the internal nodes of the tree while the gaps make up the leaves. Our goal is to construct a binary search tree such that expected cost of search is minimized. This expected cost of search is also sometimes referred to as the expected path length. It is formally defined as:

$$P = \sum_{i=1}^{n} p_i \cdot (d_T(B_i) + 1) + \sum_{j=0}^{n} q_j \cdot (d_T(B_j, B_{j+1})) \tag{1.1}$$

where p_i and q_j are the probabilities of searching for key B_i or gap (B_{i-1}, B_i) respectively and $d_T(B_i)$ and $d_T(B_j, B_{j+1})$ are the depths of the internal node for B_i and the leaf for (B_j, B_{j-1}) respectively in the tree T. Note that we assume that B_0 represents $-\infty$ and B_{n+1} represents ∞. Note that we charge 1 extra to search for a key at depth l than a leaf at depth l because it requires an extra operation to confirm an internal node, whereas we do not need this confirmation if the node is a leaf. The optimal solution of Knuth uses dynamic programming and requires $\Theta(n^2)$ time, and $\Theta(n^2)$ space [43]. This solution is both time and space intensive. We will later examine an approximate solution to this problem of Güttler, Mehlhorn and Schneider (the Modified Entropy Rule) which uses $O(n^2)$ time but $O(n)$ space [32]. We will improve its worst-case expected search cost bound. While all of the aforementioned algorithms examine the problem in the RAM model, we will also examine the problem in more realistic models of memory and look at approximate solutions under these settings.

1.3 Three-Way Branching

While modern computers typically only support two-way branching, the optimum binary search tree (BST) problem proposed by Knuth uses the three-way branch model. This

model allows a single comparison operation to transfer control to three different locations.

Examples of this can be seen in FORTRAN IV which describes the arithmetic IF [20]:

```
IF (EXPR) LABEL1, LABEL2, LABEL3
```

Control is transferred to LABEL1, LABEL2 or LABEL3 if $expr < 0$, $expr = 0$, or $expr > 0$ respectively using a single comparison command. While modern programming languages scarcely use this arithmetic IF, and compilers may simply encode such expressions using multiple logical if statements, many machines in in the FORTRAN IV era did. For example, the ARM instruction set would utilize condition codes based on comparison operations which could express negative, zero, or positive values. The condition codes would then be examined to determine control flow [6]. The difference between a two-way branch model is significant and can be seen through a simple example of searching among 3 keys.

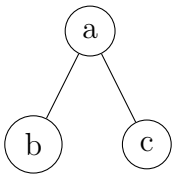


Figure 1.1: A 3 node binary tree.

Assume the probability of search for any of a, b, or c is $\frac{1}{3}$. Under the three-way branch model, this can be done using exactly one comparison by asking if the key we are searching for is strictly less than b, equal to b, or strictly greater than b and returning the correct key appropriately. Under a standard two-way branch model, this would require an extra comparison operation $\frac{2}{3}$ of the time as we can only distinguish one of the the three cases from the other two using a single two-way $<, >$, or $=$ comparison operation. We get an expected cost of 1 comparison in the three-way branch model and $\frac{5}{3}$ comparisons in the two-way model. In general, each comparison can reveal up to $\lg(3)$ bits of information in the three-way model, while only 1 bit per comparison can be revealed in the two-way model.

1.4 Why Study Binary Search Trees

Traditional binary search trees (without probabilities of searching for different keys) are ubiquitous in computer science with numerous applications. The basic binary search tree

has extended in a number of ways. AVL trees (named after creators AdelsonVelskii and Landis) were the first form of self-balancing binary search trees introduced [1]. This type of tree was invented by the pair in 1963 and maintains a height of $O(\lg(n))$ (where n is the number of nodes in the tree) during insertions and deletions (both of which take $O(\lg(n))$ time). Improved self-balancing binary search trees followed in the form of Symmetric binary B-trees by R. Bayer in 1972 [8]. These are commonly referred to as red-black trees, a term coined by Guibas, Sedgewick and Robert in 1978 [31]. Allen and Munro followed with self-organizing binary search trees and examined a move-to-root heuristic, demonstrating that its expected search time was within a constant of the optimal static binary tree [4]. The famous splay trees of Sleator and Tarjan followed in 1985 [50]. Tango trees were invented in 2007 by Demaine et al. and provided the first $O(\lg \lg n)$-competitive binary tree [19]. Here, $O(\lg \lg n)$-competitive means that the tango tree does at most $O(\lg \lg n)$ times more work (pointer movements and rotations) than an optimal offline tree. B trees are among the most commonly used binary tree variant and were invented in 1970 by R. Bayer and McCreight [9].

BST's and their extensions are integral to a large number of applications. For example, a BST variant known as the binary space partition is a method for recursively subdividing space in order to store information in an easily accessible way. It is used extensively in 3D graphics [49, 48]. Binary tries are similar to binary trees, but only store values for leaf nodes. Binary tries are routinely used in routers and IP lookup data structures [51]. Another example can be seen in the C++ `std::map` data structure, which is usually implemented using a red-black tree (another extension of binary search trees) in order to store its key-value pairs [17]. Finally, syntax trees (trees used in the parsing of various programming languages) are created using binary (and more complicated) tree structures. These trees are used in the parsing of written code during compilation [46]. Optimum binary search trees themselves were used in a variety of text indexing applications (like the KWIC index of the late 1960s [33]), especially around the time when they first appeared in the literature.

1.5 Overview

In Chapter 2, we review previous work done in the areas of binary search trees, multiway trees, alphabetic codes and various models of external memory. In Chapter 3, we re-examine the modified entropy rule of Güttler, Mehlhorn and Schneider [32]. This is an $\Theta(n^2)$ time, $\Theta(n)$ space, algorithm for approximating the optimum binary search tree

problem in the RAM model. The method works very well in practice, and the group had great experimental results, but unfortunately they could not bound the worst case expected cost as well as they would have hoped. While simpler solutions like the *Min-max* of P. Bayer [7] and *Weight Balanced* technique of Knuth [43] have worst case costs of at most $H + 2$, the trio's modified entropy technique was only shown to have a worst case expected search cost of at most $c \cdot H + 2$ where $c \approx 1.08$ [7, 32]. We provide a new argument of the modified entropy rule's worst case expected search cost and show that it is within an additive factor of entropy: at worst $H + 4$. In Chapter 4, we move on to external memory models, examining the optimum binary search tree problem under the Hierarchical Memory Model of Aggarwal et al. [2]. We provide two algorithms which run in $O(n)$ time and bound their worst case expected costs. We show that the solutions provided both give a direct improvement over a solution of Thite provided under the related HMM_2 model [52]. In Chapter 5, we consider a variant of the optimum binary search tree problem (in the RAM model) where the set of probabilities given are from an unordered multiset. We show that for a multiset with n probabilities, a simple greedy algorithm is within $\frac{n+1}{2n}$ of optimal. Finally, in Chapter 6, we summarize our findings and discuss several problems which remain open.

Chapter 2

Background and Related Work

In this chapter we provide an overview of relevant work on binary search trees, alphabetic codes, multiway search trees and models of external memory.

2.1 Binary Search Trees

In 1971, C. Gotlieb and Walker gave an approximate solution to the optimum binary search tree problem [57]. Knuth shortly thereafter gave the first optimal solution [43]. Knuth's optimal solution requires $O(n^2)$ time and space which is too costly in many situations. Several others have since examined the approximate version of the problem. While unable to bound an approximate algorithm within a constant of the optimal solution, many authors have been able to bound the cost based on the entropy of the distribution of probabilities, H. Specifically,

$$H = \sum_{i=1}^{n} p_i \cdot \lg(\frac{1}{p_i}) + \sum_{j=0}^{n} q_i \cdot \lg(\frac{1}{q_j}).$$

In 1975, P. Bayer showed that

$$H - \lg H - (\lg e - 1) \leq C_{Opt} \leq C_{WB}, C_{MM} \leq H + 2$$

where C_{Opt}, C_{WB}, and C_{MM} are costs for the optimal solution, as well as weight-balanced method of Knuth [43] and min-max heuristic of P. Bayer [7]. Weight-balanced and min-max cost heuristics are greedy and require both $O(n)$ time and $O(n)$ space to run with

the $O(n)$ implementations due to Fredman [22]. These greedy heuristics use a top-down approach where the tree root is selected from among the n keys, and we recurse in both the left and right subtrees. Let $P_L(B_i)$ and $P_R(B_i)$ represent the probabilities of searching for a key before or after key B_i respectively. The Weight-balanced approach, makes this greedy root selection by picking the root B_i such that $|P_L(B_i) - P_R(B_i)|$ is minimized. The min-max heuristic selects the root B_i such with minimum $\max(P_L(B_i), P_R(B_i))$. In 1980, Güttler, Mehlhorn and Schneider presented a new heuristic, the modified entropy rule [32] which built upon the ideas of Horibe [35]. The Entropy Rule greedily selects B_i as the root such that

$$H(P_L(B_i), p_i, P_R(B_i)) = P_L(B_i) \cdot \lg(\frac{1}{P_L(B_i)}) + p_i \cdot \lg(\frac{1}{p_i}) + P_R(B_i) \cdot \lg(\frac{1}{P_R(B_i)})$$

is maximized. As discussed in Chapter 3, this was modified to improve its performance. Güttler, Mehlhorn and Schneider gave empirical evidence that the modified heuristic outperformed others [32]. While the heuristic took $O(n^2)$ time, it only required $O(n)$ space, a huge savings over the optimal solution. However, they were unable to prove that the cost of the modified entropy rule $C_{ME} \leq H + 2$ (unlike previous weight-balanced and min-max heuristics) and settled with $C_{ME} \leq c_1 \cdot H + 2$ where $c_1 = \frac{1}{H(\frac{1}{3}, \frac{2}{3})} \approx 1.08$. We re-examine this method and provide a new bound of $H + 4$ in Chapter 3. In 1993, De Prisco and De Santis presented a new heuristic for constructing a near-optimum binary search tree [18]. The method is discussed in more detail in section 4.6 and has an upper bounded cost of at most $H + 1 - q_0 - q_n + q_{max}$ where q_{max} is the maximum weight leaf node. This method was later updated by Bose and Douïeb (and is also discussed in section 4.6) to have a worst case cost of [15]

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}.$$

Here, $m' = \max(2n - 3P, P) - 1 \geq \frac{n}{2} - 1$ where P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves (gaps) and, $pq_{\text{rank}[i]}$ is the i^{th} smallest access probability among all keys and gaps except q_0 and q_n.

2.2 Alphabetic Codes

Determining the optimum alphabetic coding is an important related problem relevant later in this thesis. Given a set of n keys (keys $B_1, ..., B_n$) with various probabilities, we wish

to build a binary search tree where every internal node has two children, and the n keys described are the leaves. We wish to build the tree with the minimum expected search cost. The expected search cost is

$$\sum_{i=1}^{i=n} p_i \cdot d_T(B_i)$$

where p_i is the probability of searching for key B_i and $d_T(B_i)$ is the depth of the leaf representing the key B_i in the tree T. The alphabetic ordering of the leaves must be maintained. This is the same as the binary search tree problem with all internal node weights zero.

In 1952, Huffman developed the well known Huffman tree, which solved the same problem without a lexicographic ordering constraint on leaves [40]. Gilbert and Moore were the first to examine the problem with the added alphabetic constraint (alphabetical codes) and developed a $O(n^3)$ algorithm which solved the problem optimally [26]. Hu and Tucker gave a $O(n^2)$ time and $O(n)$ space algorithm in 1971 [39] which was improved by Knuth to take only $O(n \lg n)$ time and $O(n)$ space in 1973 [42]. The original proof of Hu and Tucker was extremely complicated, but was later simplified by Hu [36] and Hu et al. [38]. Garsia and Wachs gave an independent $O(n \lg n)$ time, $O(n)$ space algorithm in 1977 [25]. This new algorithm by Garsia and Wachs was shown to be equivalent to the Hu and Tucker algorithm in 1982 by Hu [37] and also went through a proof simplification [41] by Kingston in 1988.

In 1991, Yeung proposed an approximate solution which solved the problem in $O(n)$ time and space [59]. The algorithm produced a tree with worst case cost $H + 2 - p_1 - p_n$. This algorithm was later improved by De Prisco and De Santis who created an $O(n)$ time algorithm which has a worst case cost of $H + 1 - p_1 - p_n + p_{max}$ [18]. The method was improved one more time by Bose and Douïeb who improved upon Yeung's method by decreasing the bound by $\sum_{i=0}^{m} p_{\text{rank}[i]}$ where $m = \max(n - 3P, P) - 1 \geq \frac{n}{4} - 1$, P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves and $p_{\text{rank}[i]}$ is the i^{th} smallest access probability among all leaves except p_1 and p_n [15]. Replacing Yeung's method with the improved algorithm of Bose and Douïeb in the De Prisco and De Santis algorithm gave the tightest bound seen so far of

$$H + 2 - p_1 - p_n - \sum_{i=0}^{m} p_{\text{rank}[i]}.$$

2.3 Multiway Trees

Another related problem is the static k-ary or multiway search tree problem. It is similar to the optimum binary search tree problem with the added option that up to $k-1$ keys can be placed into a single node, and the cost of search within a node is constant. Multiway search trees maintain an ordering property similar to that of traditional binary search trees. Each key in every page g that isn't the root page must have its keys lie between some keys l and l', two keys located in $g's$ parent. Each internal node of the k-ary tree contains at least one and at most $k-1$ keys while a leaf node contains no keys and represents searching for an item in a gap between keys. Successful searches end in an internal node while unsuccessful searches end in one of the $n+1$ leaves of the tree. The cost of search is the average path depth which is defined as:

$$\sum_{i=1}^{n} p_i(d_T(B_i)+1) + \sum_{j=0}^{n} q_j(d_T((B_{i-1}, B_i)))$$

where B_i's represent successful search keys with probabilities p_i, pairs (B_{i-1}, B_i) (with probabilities q_{i-1}) represent gaps and $d_T(B_i)$ or $d_T((B_{i-1}, B_i))$ represent the depth of keys or gaps respectively in the tree T.

Vishnavi et al. [53], and Gotlieb [29] in 1980 and 1981, respectively, independently solved the problem optimally in $O(k \cdot n^3)$ time. In a slightly modified B-tree model (every leaf has the same depth, every internal node is at least half full), Becker's 1994 work gave a $O(kn^{\alpha})$ time algorithm where $\alpha = 2+log2/log(k+1)$ [10]. Later, in 1997, Becker proposed an $O(Dkn)$ time algorithm where D is the height of the resulting tree [11]. The algorithm did not produce an optimal tree but was thought to be empirically close despite having no strong upper bound. In 2009, Bose and Douïeb gave both an upper and lower bound on the optimal search tree in terms of the entropy of the probability distribution as well as an $O(n)$ time algorithm to build a near-optimal tree [15]. Their bounds of

$$\frac{H}{\lg(2k-1)} \leq P_{OPT} \leq P_T \leq \frac{H}{\lg k} + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}$$

are discussed in more detail in section 4.3 of this paper. Here, $m = max(n-3P, P) - 1 \geq \frac{n}{4} - 1$. P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the gaps. Moreover, $q_{\text{rank}[i]}$ is the i^{th} smallest access probability among all leaves (gaps) except q_0 and q_n.

2.4 Memory Models

In the typical RAM model, we assume that all reads and writes from memory take a constant amount of time. While this is a valid assumption in many situations (both in the context of theory and programming) when dealing with very large data sets, this is simply not the case. A typical computer has a memory hierarchy with CPU registers, various levels of cache, RAM, SSD and/or hard drives. Each of these memory levels has increasing size but decreasing I/O speed. Typically, the difference between the levels is dramatic (reading from disk takes roughly a million times longer than accessing a CPU register) [54]. Moreover, many memory hierarchies allow blocks of memory to be moved quickly after a single key or cache line has been accessed. It is thus possible to take advantage of the locality of data during computations. Moreover, it is imperative to consider memory I/O speeds, especially when the dataset being worked on does not fit on internal memory. *External memory* algorithms and data structures refer to those methods and structures which explicitly manage data placement and movement [54]. Various authors have created models to properly reflect the performance of such algorithms and data structures, and we consider the optimum binary search tree problem under such a model.

In Chapter 4, we discuss the optimum binary search tree problem under the 1987 Hierarchical Memory Model of Aggarwal et al. [2]. The model is described thoroughly in section 4.1, but essentially provides an alternative to the classic RAM model. It simulates a memory hierarchy with various memory sizes and different access times for each type of memory. The model does have its shortcomings though as it does not provide us with the ability to move blocks of memory between these different memory types (as in a typical computer). The HMM model was later extended to the Hierarchical Memory with Block Transfer Model of Aggarwal, Chandra, and Snir [3]. This model provides a less artificial setting by allowing for contiguous blocks of memory to be copied from one location to another for a cheaper price. The cost of this copy equal to the cost to access the most expensive location being copied (to or from), plus the size of the block.

As explained in the survey of Vitter [54], several other models of external memory have followed. Work has been done to consider models with parallelism. Vitter and Shriver [56] built upon the HMM and HMBTM models of Aggarwal et al. [2, 3] in order to allow parallelism. This updated model connects P parallel memory hierarchies at their base memory levels. In 1994, Alpern et al. introduced the Uniform Memory Hierarchy (UMH) [5]. The UMH considers block sizes and bandwidths between memory levels, and allows for simultaneous transfer between pairs of memory levels. The UMH was considered with additional parallelization by Vitter and Nodine [55].

Cache-oblivious algorithms were introduced by Frigo, Leiserson, Prokop and Ramachan-

dran in 1999 [23]. The model used is the ideal-cache model which has a two-level memory hierarchy. The internal memory (named cache) has Z words and the main memory is arbitrarily large. The cache is divided into cache lines of size L and it is assumed that $Z = \Omega(L^2)$ (the tall cache assumption). Frigo et al. examined the fast fourier transform and matrix multiplication under this model. Many others have since used this model and examined problems in a cache-oblivious setting such as the cache-oblivious b-trees of Bender et al. [12], the funnel heap of Brodal et al. [16], or the locality preserving cache-oblivious dynamic dictionary of Bender et al. [13].

A complete and thorough explanation of memory hierarchies (especially those considered before cache-oblivious settings) can be found in the survey of Vitter [54].

Chapter 3

An Improved Bound for the Modified Minimum Entropy Heuristic

In this chapter we show that the Modified Minimum Entropy Heuristic of Güttler, Mehlhorn and Schneider [32] is within an additive factor of entropy: at worst $H + 4$. The previous bound was best upper bound was $c_1 \cdot H + 2$ where $c_1 = \frac{1}{H(\frac{1}{3}, \frac{2}{3})} \approx 1.08$.

3.1 Preliminaries

Recall equation 1.1, $H = \sum_{i=1}^{n} p_i \cdot \lg(\frac{1}{p_i}) + \sum_{j=0}^{n} q_i \cdot \lg(\frac{1}{q_j})$. We also use

$$H(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i \cdot \lg(\frac{1}{x_i})$$

to describe the entropy of any probability distribution $(x_1, x_2, ..., x_n)$. For subtree t, we let

$$p_t = \sum_{i:B_i \in t} p_i + \sum_{i:(B_i, B_{i+1}) \in t} q_i$$

be its total probability (the sum of the probability of all nodes within the subtree). $P_L(B_i)$ and $P_R(B_i)$ are probabilities of searching lexicographically before or after (respectively) key B_i. $P_L(B_i, B_{i+1})$ and $P_R(B_i, B_{i+1})$ are probabilities of searching lexicographically before (or equal to) B_i and after (or equal to) B_{i+1} respectively. For a subtree t, $P_L^t(B_i)$ and

$P_R^t(B_i)$ describe the normalized probabilities of searching for a key to the left or right of B_i within t, and $P_L^t(B_i, B_{i+1})$ and $P_R^t(B_i, B_{i+1})$ have analogous definitions. We let

$$E_t = H(P_L^t(B_i), \frac{p_i}{p_t}, P_R^t(B_i)) \tag{3.1}$$

be the local entropy of a subtree t rooted at key B_i.

3.2 The Modified Entropy Rule

We first describe the entropy rule originally by Horibe [35] for greedy root selection then explain how it was modified in [32]. For a subtree t with probability p_t, the entropy rule greedily chooses the key B_i as the root such that $H(P_L^t(B_i), \frac{p_i}{p_t}, P_R^t(B_i))$ is maximized. While this rule behaves quite well in practice, certain cases cause it to have poor performance (refer to Figure 3.1).

(a) Entropy rule tree: $C = \frac{8}{5}$ (b) Modified entropy rule tree: $C = \frac{7}{5}$

Figure 3.1: Comparison of entropy and modified entropy rule heuristics

Figure 3.1 demonstrates the shortcomings of the entropy rule heuristic. Given the probability set $\{q_0 = \frac{1}{5}, p_1 = \frac{1}{5}, q_1 = 0, p_2 = 0, q_3 = \frac{3}{5}\}$ the entropy rule will mistakenly choose key B_1 as the root while selecting B_2 as the root produces a better tree. This mistake is remedied in the modified entropy rule of Güttler, Mehlhorn and Schneider [32]. The modified entropy heuristic chooses the root in one of the following three ways:

a) If there exists key B_i such that $\frac{p_i}{p_t} > \max(P_L^t(B_i), P_R^t(B_i))$ we always select B_i as the root.

b) If there exists a gap (B_i, B_{i+1}) such that $\frac{q_i}{p_t} > max(P_L^t(B_i, B_{i+1}), P_R^t(B_i, B_{i+1}))$ then we select the root from among B_i and B_{i+1}. B_i is chosen if $P_L^t(B_i, B_{i+1}) > P_R^t(B_i, B_{i+1})$ and B_{i+1} is chosen otherwise.

c) Otherwise, B_i is selected such that $H(P_L^t(B_i), \frac{p_i}{p_t}, P_R^t(B_i))$ is maximized (as in the original entropy rule).

The approach proposed by Güttler, Mehlhorn and Schneider takes $O(n^2)$ time in the worst case and $O(n)$ space.

3.3 Modified Entropy is Within 4 of Entropy

First, we review a quick Lemma about entropy (nearly identical to Lemma 2.3 in [7]).

Lemma 3.3.1. *If $x \leq \frac{1}{2}$ then $H(x, 1-x) \geq 2x$.*

Proof. We refer the reader to Gallager's 1968 work [24]. □

Next, we describe a Lemma which breaks our choice of root in the greedy modified entropy heuristic into one of three cases (not to be confused with the three *rules* used in section 3.2).

Lemma 3.3.2. *When using the modified entropy rule chooses the root B_r of a subtree t with total probability p_t, one of the following three cases must occur:*

Case 1) $E_t \geq 1 - 2\frac{p_r}{p_t}$

Case 2) There exists gap(B_i, B_{i+1}) *such that* $\frac{q_i}{p_t} > max(P_L^t(B_i, B_{i+1}), P_R^t(B_i, B_{i+1}))$

Case 3) $\max(P_L^t(B_r), P_R^t(B_r)) < \frac{4}{5}$

Proof. At a high level, we first show that *Rule a)* from section 3.2 implies *Case 1*. We also show that if there exists B_i such that $P_L^t(B_i) \leq \frac{1}{2}$ and $P_R^t(B_i) \leq \frac{1}{2}$, but cannot apply *Rule a)*, then we still have *Case 1*. Assuming that neither of the two aforementioned conditions occur, we must have that there exists some gap (B_i, B_{i+1}) spanning the middle of the data set. Given this condition, we show that if *Case 2* does not occur (i.e. we cannot use *Rule b)* of section 3.2) then *Case 3* must occur, completing the proof.

Rule a)* $\implies$ *Case 1

First, suppose there exists some p_i such that $\frac{p_i}{p_t} > \max(P_L^t(B_i), P_R^t(B_i))$. By the *Rule a)* of section 3.2, it must be selected as the root and thus $r = i$. Moreover, both $P_L^t(B_i)$ and $P_R^t(B_i))$ must be less than one half. Thus, using Lemma 3.3.1 we have:

$$\begin{aligned} E_t &\geq H(max(P_L^t(p_i), P_R^t(p_i)), 1 - max(P_L^t(p_i), P_R^t(p_i)) \\ &\geq 2 \cdot \max(P_L^t(p_i), P_R^t(p_i)) \\ &\geq 1 - \frac{p_i}{p_t} \\ &\geq 1 - 2\frac{p_i}{p_t} = 1 - 2\frac{p_r}{p_t} \end{aligned}$$

as required.

B_i spans middle* $\implies$ *Case 1

If we do not have some p_i such that $\frac{p_i}{p_t} > \max(P_L^t(B_i), P_R^t(B_i))$ but do have some B_i such that $P_L^t(B_i) \leq \frac{1}{2}$ and $P_R^t(B_i) \leq \frac{1}{2}$ then we must use *Rule c)* of section 3.2. We then must have that:

$$\begin{aligned} E_t &\geq H(P_L^t(B_i), \frac{B_i}{p_t}, P_R^t(B_i)) \text{ and} \\ 0 &\leq P_L^t(B_i) \leq 0.5 \text{ and} \\ 0 &\leq \frac{p_i}{p_t} \leq 0.5 \text{ and} \\ 0 &\leq P_R^t(B_i) \leq 0.5 \end{aligned}$$

then we know that

$$H(P_L^t(p_i), \frac{p_i}{p_t}, P_R^t(p_i)) \geq H(1/2, 1/2) = 1$$

in this case. Thus, combining the above two cases, if have some B_i such that $P_L^t(B_i) \leq \frac{1}{2}$ and $P_R^t(B_i) \leq \frac{1}{2}$ then $E_t \geq 1 - 2p_r$ as required.

(NOT(B_i spans mid) AND NOT(Case 1) AND NOT(Case 2))* $\implies$ *Case 3

Otherwise, we must have some gap (B_i, B_{i+1}) spanning the middle of the data set (i.e. $P_L^t(B_i, B_{i+1}) < \frac{1}{2}$ and $P_R^t(B_i, B_{i+1}) < \frac{1}{2}$). Suppose that *Case 2* does not occur (i.e. we cannot use *Rule b*): there does not exist a (B_i, B_{i+1}) such that $\frac{q_i}{p_t} > max(P_L^t(B_i, B_{i+1}), P_R^t(B_i, B_{i+1}))$.

Then, for any root r of t we have that

$$\max(P_L^t(B_r), P_R^t(B_r)) \geq \min(P_L^t(B_r), P_R^t(B_r))$$
$$\max(P_L^t(B_r), P_R^t(B_r)) \geq q_i$$

Thus, for any root r:

$$\max(P_L^t(p_r), P_R^t(p_r)) \geq 1/3$$

and by our assumption

$$\max(P_L^t(p_r), P_R^t(p_r)) < \frac{1}{2}.$$

So, as in the proof of table 3 (5.3) in [32]

$$E_t \geq H(1/3, 2/3) \approx 0.92.$$

Either *Case 1* occurs, or we have that:

$$E_t < 1 - 2\frac{p_r}{p_t}$$
$$\implies \frac{p_r}{p_t} < \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2} \approx 0.04.$$

Suppose for contradiction that $\max(P_L^t(p_r), P_R^t(p_r)) \geq \frac{4}{5}p_t$ then we have:

$$E_t \leq H(\frac{4}{5}, \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2}, \frac{1}{5} - \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2}) \approx 0.87 < 0.92 \approx H(\frac{1}{3}, \frac{2}{3}) \leq E_t$$

which is a contradiction. Thus, if we do not have *Case 1* or *Case 2* we must have *Case 3* which completes the proof.

□

Before we examine the main theorem we show a small claim.

Claim 1. $H(\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x) \geq 1 - \frac{4}{5}x^2$ *when* $0 < x < \frac{1}{2}$

Proof. In order to prove the claim, we find the minimum of

$$H(\frac{1}{2}-\frac{1}{2}x,\frac{1}{2}+\frac{1}{2}x)-(1-\frac{4}{5}x^2)$$

when $0<x<\frac{1}{2}$. To do this, we define $F(x)$ and take the derivative with respect to x.

$$\begin{aligned}
F(x) &= H(\frac{1}{2}-\frac{1}{2}x,\frac{1}{2}+\frac{1}{2}x)-(1-\frac{4}{5}(x)^2)\\
F(x) &= -(\frac{1}{2}-\frac{1}{2}x)\cdot\lg(\frac{1}{2}-\frac{1}{2}x)-(\frac{1}{2}+\frac{1}{2}x)\cdot\lg(\frac{1}{2}+\frac{1}{2}x)-(1-\frac{4}{5}x^2)\\
\implies F'(x) &= \lg(\frac{1}{2}-\frac{1}{2}x)-\lg(\frac{1}{2}+\frac{1}{2}x)+\frac{8}{5}x \text{ (with some careful manipulation)}
\end{aligned}$$

The only root occurs when $x=0$. Thus, we check when $x\to 0$ and $x\to\frac{1}{2}$. We note that:

$$\begin{aligned}
F'(x) &\xrightarrow{x\to 0} 0^+ \text{ and}\\
F'(x) &\xrightarrow{x\to\frac{1}{2}} 0.0112781>0
\end{aligned}$$

Thus, $H(\frac{1}{2}-\frac{1}{2}x,\frac{1}{2}+\frac{1}{2}x)-(1-\frac{4}{5}(x)^2)>0$ for $0<x<\frac{1}{2}$ which proves the claim. □

Theorem 3.3.3. *Let C_{ME} be the expected cost of search for a tree made by the modified entropy rule. Then*

$$C_{ME}\le H+4$$

Proof. This uses a similar style to the proof of Theorem 4.4 in [7]. We bind each E_t for each subtree of our BST on a case by case basis using the cases of Lemma 3.3.2.
If *Case 1* occurs, we obviously have that

$$E_t\ge 1-2\frac{p_r}{p_t}. \tag{3.2}$$

Note that this can only happen once for each key (a key can only be root once).

As mentioned in Lemma 3.3.2 if some B_i spans in the middle of the data set, $P_L^t(B_i)\le\frac{1}{2}$ and $P_R^t(B_i)\le\frac{1}{2}$, we can still show that *Case 1* occurs. Suppose for the remainder of the proof that there is no such middle-spanning B_i.

Let (B_m, B_{m+1}) be the unique middle gap (i.e. $P_L^t(B_m, B_{m+1}) < \frac{1}{2}$ and $P_R^t(B_m, B_{m+1}) < \frac{1}{2}$) when *Case 2* or *Case 3* occurs. When *Case 2* occurs, we know that we could select a root from among the two keys outside of our middle spanning gap, and choose the one which is closer to the middle. Thus, we have that (using Lemma 3.3.1):

$$E_t \geq H(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}, \frac{1}{2} + \frac{1}{2}\frac{q_m}{p_t}) \geq 2(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}) = 1 - \frac{q_m}{p_t}. \tag{3.3}$$

Note that by the definition of the *Rule c)* of section 3.2, when this occurs, (B_m, B_{m+1}) must be a leaf of depth at most 2. Thus, this condition can only happen twice for each (B_m, B_{m+1}) gap.

When neither *Case 1* nor *Case 2* occur (and we have a (B_m, B_{m+1}) spanning the middle) we must have *Case 3*. This gives us

$$E_t \geq H(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}, \frac{1}{2} + \frac{1}{2}\frac{q_m}{p_t}).$$

We again apply Claim 1 and get

$$E_t \geq 1 - \frac{4}{5}(\frac{q_m}{p_t})^2. \tag{3.4}$$

As in [7] we define a b_t for each subtree t as follows. We want to have a value for b_t such that $E_t \geq 1 - \frac{b_t}{p_t}$ in all cases. Using *Cases 1,2*, and *3* are their respective equations 3.2, 3.3, and 3.4 we do just that:
Let $b_t = 2 \cdot p_r$ when *Case 1* occurs. B_r is the root of b_t.
Let $b_t = 2 \cdot q_m$ when *Case 2* occurs. (B_m, B_{m+1}) is middle gap of b_t.
Let $b_t = \frac{q_m^2}{p_t}$ when *Case 3* occurs. (B_m, B_{m+1}) is middle gap of b_t.

Note that, in 1975 P. Bayer [7] showed that the cost C of our tree could be defined as (**Lemma 2.3**)

$$C = \sum_{t \in S_T} p_t$$

and the entropy could be calculated by

$$H = \sum_{t \in S_T} p_t \cdot E_t$$

where S_T is the set of all subtrees of our tree T.

Thus, by subbing in $E_t \geq 1 - \frac{b_t}{p_t}$ and rearranging we get:

$$H = \sum_{t \in S_T} p_t \cdot E_t \geq \sum_{t \in S_T} p_t - \sum_{t \in S_T} b_t = C - \sum_{t \in S_T} b_t$$
$$\implies C \leq H + \sum_{t \in S_T} b_t$$

As mentioned above, *Case 1* and *Case 2* can only occur once and twice respectively for any potential root B_r or gap (B_m, B_{m+1}). *Case 3* however, can occur many times for a gap (B_m, B_{m+1}). Each time it occurs though, $\frac{q_m}{p_t}$ must increase by a factor of at least $\frac{5}{4}$ since $\max(P_L^t(B_r), P_R^t(p_r)) < \frac{4}{5}$ for the root B_r of the subtree by *Case 3)* of Lemma 3.3.2. Moreover, if $\frac{q_m}{p_t} > \frac{1}{2}$ then we will have *Case 2*. Let S_m be the set of all subtrees t for which (B_m, B_{m+1}) is the middle gap and *Case 3* only applies. We have that

$$C \leq H + \sum_{t \in S_T} b_t = H + 2\sum_{r=1}^{n} p_r + 2\sum_{m=0}^{n} q_m + \sum_{m=0}^{n} \sum_{t \in S_m} \frac{4}{5} \frac{q_m^2}{p_t}$$

By factoring out q_m and examining only cases up to $\frac{q_m}{p_t} = \frac{1}{2}$ (since otherwise *Case 2* will occur) we get:

$$\begin{aligned} C &\leq H + 2 + \sum_{m=0}^{n} (\frac{4}{5} \cdot q_m) \sum_{x=0}^{\infty} \frac{1}{2} \cdot (\frac{4}{5})^x \\ &= H + 2 + \sum_{m=0}^{n} \frac{4}{5} \cdot q_m \cdot \frac{1}{2} \cdot (\frac{1}{1 - \frac{4}{5}}) \text{ (geometric series)} \\ &= H + 2 + 2 \cdot \sum_{m=0}^{n} q_m \\ &\leq H + 4. \end{aligned}$$

□

It remains open as to whether or not this bound is tight. We conjecture that it is likely the case that the worst case bound is in fact $H + 2$ as no cases with worse behaviour are known.

Chapter 4

Approximate Binary Search in the Hierarchical Memory Model

In this chapter we examine the optimum BST problem under the Hierarchical Memory Model (HMM). We provide two approximate solutions, bound their worst case expected costs via entropy, and show improvement over a previous solution in the related HMM_2 model.

4.1 The Hierarchical Memory Model

The HMM was proposed in 1987 by Aggarwal et al. as an alternative to the classic RAM model [2]. It was intended to better model the multiple levels of the memory hierarchy. The model has an unlimited number of registers, $R_1, R_2, ...$ each with its own location in memory (a positive integer). In the first version of the model, accessing a register at memory location x_i takes $\lceil \lg(x_i) \rceil$ time. Thus, computing $f(a_1, a_2, ..., a_n)$ takes $\sum_{i=1}^{n} \lceil \lg(location(a_i)) \rceil$ time. The original paper also considered arbitrary cost functions $f(x)$. We will use the cost function as was explained in Thite's thesis [52]. Here, $\mu(a)$ is the cost of accessing memory location a. We have a set of memory sizes $m_1, m_2, ..., m_l$ which are monotonically increasing. Each memory level has a finite size except m_l which we assume has infinite size. Each memory level has an associated cost of access $c_1, c_2, ..., c_l$. We assume that $c_1 < c_2 < ... < c_l$. The cost of accessing a memory location a is given by

$$\mu(a) = c_i \text{ if } \sum_{j=1}^{i-1} m_j < a \leq \sum_{j=1}^{i} m_j. \tag{4.1}$$

More specifically, we can think of c_i as the entire cost of moving a element in m_i recursively up the memory hierarchy and finally accessing it. While there are interesting problems in more sophisticated models, it is beyond the scope of this work to examine locality optimizations under a more complex model in this recursive movement of items up the hierarchy.

Thite notes that typical memory hierarchies have decreasing sizes for faster memory levels (moving *up* the memory hierarchy). We make the same assumption:

$$m_1 < m_2 < ... < m_l.$$

Unlike Thite, we also explicitly assume that successive memory level sizes divide one another evenly:

$$\forall i \in \{1, 2, ..., l-1\} m_i \mid m_{i+1}.$$

4.2 Thite's Optimum Binary Search Trees on the HMM Model

Thite's thesis provides solutions to several problems in the HMM and the related HMM_2 models [52]. He first provides an optimal solution to the following problem (known as **Problem 5** in the work). Note that we have made slight modifications in order to maintain correct notation throughout this work:

> ***Problem 1 [Optimum BST Under HMM].*** *Suppose we are given a set of n ordered keys $B_1, B_2, ..., B_n$ with associated probabilities of search $p_1, p_2, ..., p_n$, as well as $n+1$ ranges or gaps $(B_0, B_1), (B_1, B_2), ..., (B_{n-1}, B_n), (B_n, B_{n+1})$ with associated probabilities of search $q_0, q_1, ..., q_n$. The problem is to construct a binary search tree T over the set of keys and gaps (keys must be internal nodes, and gaps must be leaves) and compute a memory assignment function $\phi : V(T) \rightarrow 1, 2, ..., n$ that assigns nodes of T to memory locations such that the expected cost of a search is minimized under the HMM model* [52].

Note that we assume that B_0 represents $-\infty$ and B_{n+1} represents ∞.

Thite provided three separate optimum solutions to the problem described; **Parts**, **Trunks**, and **Split**. Here, h is the minimum memory level such that all n keys can fit on

memories of height at most h. More specifically h is defined as

$$\min(h \in \{1, .., l\}) : n \leq \sum_{i=1}^{h} m_i$$

We now give a high-level overview the the three solutions of Thite. For a more detailed explanation, please refer to his thesis [52].

Parts is a bottom up dynamic programming algorithm which constructs optimum subtrees $T^*_{i,j}$ for each i, j, $1 \leq i \leq j \leq n$ using only the first $j-i+1$ memory locations. For each choice of root $B(k)$ where $k \in \{i, ..., j\}$ all possible memory assignments are examined. Thite claims a running time of $O(\frac{2^{h-1}}{(h-1)!} \cdot n^{2h+1})$ for the algorithm.

Trunks is an algorithm that, like Parts, uses dynamic programming to build optimal subtrees over larger and larger sets of keys. For each range i, j, $1 \leq i \leq j \leq n$, and for each possible root $B(k)$, they examine all s such that $s \leq j-i+1$ and consider placing s nodes in the first $h-1$ levels of memory, and the remaining $j-i-1-s$ nodes in memory level h. Thite claims that the algorithm takes $O(\frac{2^{n-m_h} \cdot (n-m_h+h)^{n-m_h} \cdot n^3}{(h-2)!})$ time. Moreover, the algorithm should work well in practice when $n - m_h$ and h are small (i.e. a short memory hierarchy with a very large biggest memory size, typical in a modern computer).

Split is a top-down algorithm for solving the problem when there are n levels in the memory hierarchy. Thite claims a running time of $O(2^n)$. We note that this running time result is dubious as Thite's explanation simply states that a root is chosen by examining each of the 2^{n-1} ways of partitioning memory locations between the left and right subtrees and recursing on each side. It is unclear how the potential roots are compared, and we cannot think of a way to select the correct root in this fashion without an exponential running time. A proof of correctness of algorithm was unfortunately omitted from Thite's work.

In the following sections, we provide two approximate solutions to this problem that run in time $O(n)$ and provide an upper bound on their expected search costs.

Thite also considered the same problem under the related HMM_2 model. This model assumes there are simply two levels of memory of size m_1 and m_2 with costs of access c_1 and c_2 where $c_1 < c_2$. Thite provides an optimal solution to this problem (named **TwoLevel**) he claims runs in time $O(n^5)$. TwoLevel has two phases and in the first phase, it uses an algorithm similar to that of Knuth, solving for all subtrees that will fit on a single memory level (i.e. solves for all ranges $[i, j]$ such that $j - i + 1 \leq \max(m_1, m_2)$).

Specifically, it creates arrays $C[i,j]$ and $R[i,j]$ where C and R are the optimal tree cost (using a uniform cost model), and root selection of optimal tree over $[i,j]$. TwoLevel phase two utilizes algorithm Parts and uses dynamic programming to compute $c(i,j,n_1,n_2)$ and $r(i,j,n_1,n_2)$ which are the optimal tree and optimal tree root choices using the HMM_2 cost model using keys $[i,j]$ with n_1 keys in memory M_1 and n_2 in memory M_2. c and r are computed using dynamic programming. Thite claims the algorithm runs in $o(n^5)$, if $m_1 \in o(n)$, and in $O(n^4)$ if $m_1 \in O(1)$. He also gives an $O(n \lg n)$ time approximate solution with an upper bounded expected search cost of $c_2(H+1)$. This algorithm utilizes a $O(n)$ BST approximation algorithm of Mehlhorn [47] and an $O(n \lg n)$ greedy scheme for placing the BST tree into memory. The solution we provide under the HMM model also gives an improvement over Thite's approximate algorithm in both running time and expected cost under the HMM_2 model.

4.3 Efficient Near-Optimal Multiway Trees of Bose and Douïeb

In order to obtain a good approximate solution to the optimum BST problem under the HMM model, we use the multiway search tree construction algorithm of Bose and Douïeb [15]. In 2009, they devised a new method with linear running time (independent of the size of a node in tree) and with the best expected cost to date. They were able to prove that:

$$\frac{H}{\lg(2k-1)} \leq P_{OPT} \leq P_T \leq \frac{H}{\lg k} + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}.$$

Here, H is the entropy of the probability distribution, P_{OPT} is the average path-length (expected cost of search) in the optimal tree, P_T is the average path length of the tree built using their algorithm and $m = max(n-3P, P) - 1 \geq \frac{n}{4} - 1$. P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the gaps. Moreover, $q_{\text{rank}[i]}$ is the i^{th} smallest access probability among all leaves (gaps) except q_0 and q_n. Finally, k is the size of a node in the tree (the number of keys that fit inside an internal node). Each node will have at most $k+1$ children.

As described in the section 2.3, in the multiway search tree problem we are given n ordered keys with weights $p_0, ..., p_n$ as well as $n+1$ weights of unsuccessful searches $q_0, ..., q_n$. The goal is to build a minimum cost tree where k keys fit inside a single internal node. A

single gap will fit in each leaf node.

We refer readers to the paper [15] for a more detailed explanation, but we give an overview here. The algorithm proceeds in three steps:

1. An examination of the peaks and valleys of the probability distribution of the leaf weights is used to redistribute weights from leaves to internal nodes.
2. This new probability distribution is made into a k-ary tree using a greedy recursive algorithm. The algorithm recursively chooses the $l \leq k-1$ elements to go in the root node such that each child's subtree will have probability of access of at most $\frac{1}{k-1}$. This completed k-ary tree is called the *internal key tree*.
3. Leaf nodes are reattached to the *internal key tree*.

Their algorithm's design allows the authors to bound the depth of keys and gaps by their associated probabilities. This ultimately allows them to achieve the bounds on expected cost of search that they have described.

4.4 Algorithm ApproxMWPaging

In this section we provide an algorithm for creating a BST and subsequently packing it into memory. The algorithm first uses the multiway search tree construction algorithm of Bose and Douïeb as a subroutine. This multiway search tree is converted into a BST, then packed into the memory hierarchy. Note that we make the explicit assumption that we are given the keys in sorted order as input.

First, we describe how we convert a multiway search tree to a binary search tree. For the sake of clarity, we will call what are typically known as *nodes* of the multiway tree *pages*. This represents how various items of our search tree will fit onto pages of our memory hierarchy. We maintain the notion of calling individual items *keys*.

Lemma 4.4.1. *Given a multiway tree T' with page size k (a k+1-ary tree) and n keys, and $n+1$ gaps and an associated probability distribution as in Knuth's original optimum binary search tree problem, we can create a BST T where each key in a given page $g \in T'$ forms a connected component in T in $O(n)$ time.*

Proof. For each page g, we create a complete BST B over its keys (using the sorted order of all $2n+1$ keys and gaps). We create an ordering over all potential locations where *additional* keys could be added to this small BST from left to right. All keys in all descendant pages of a page g in a specific subtree rooted at a child of g will lie in a specific range. There are at most $k+1$ of these ranges (since our page has at most k keys). These ranges precisely correspond to the at most $k+1$ locations where a new child key could be added. We order these locations from left to right and attach root keys from the newly created BST's of each of the ordered (left to right) child of g. Leaf pages with a single gap are attached in a similar fashion. These are all valid connections since each child of g has keys in these correct ranges, and combining BST's in this fashion produces a valid BST. We create a complete BST in each page in $O(k)$ time (of which there are at most $O(\frac{n}{k})$ such pages), and make $O(n)$ new parent child connections, giving us total time $O(n)$. □

In order to obtain a good approximate solution to the optimum BST problem under the HMM model, we do the following:

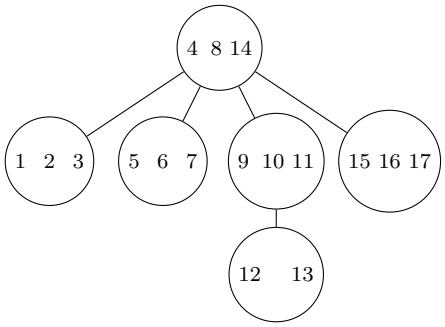
1. First, we create a multiway tree T' using the algorithm of Bose and Douïeb. This takes $O(n)$ time with our page size equal to m_1 (the smallest level of our memory hierarchy) [15].

2. Inside each page (node of the multiway tree T'), we create a balanced binary search tree (ignoring weights). We use a simple greedy approach where we sort the keys, then recursively select the middle key as the root. We call each of these T'_k for $k \in 1, ..., \lceil n/m_1 \rceil$. This takes $O(n)$ by Lemma 4.4.1.

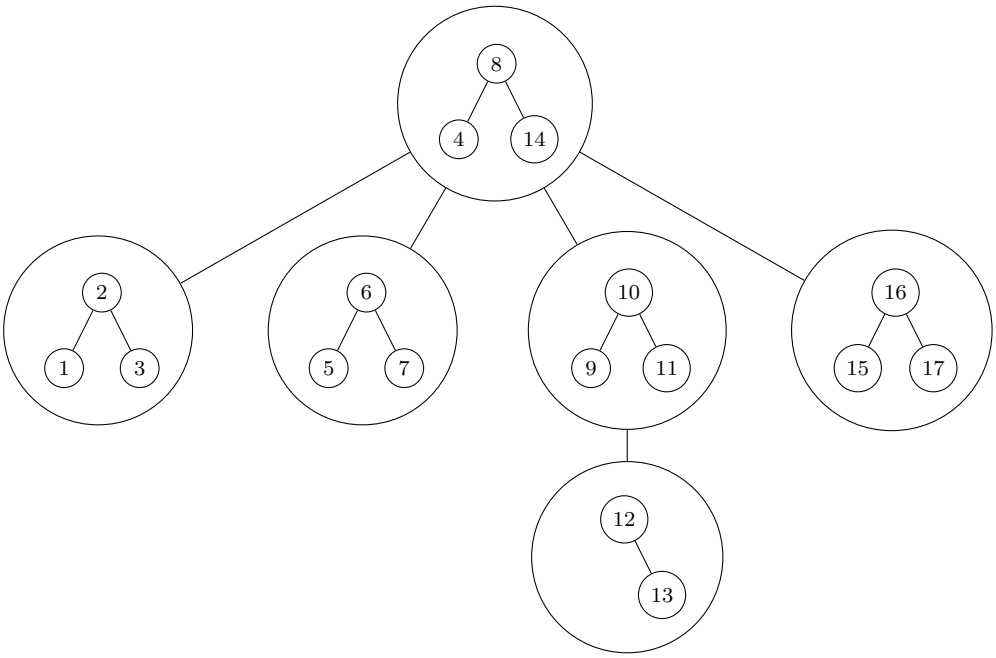
3. In order to make this into a proper binary search tree, we must connect the $O(n/m_1)$ BST's we have made as described in Lemma 4.4.1. From T', we create a BST T. This takes $O(n)$ time.

4. We pack keys into memory in a breadth first search order of T starting from the root. This takes $O(n)$ time.

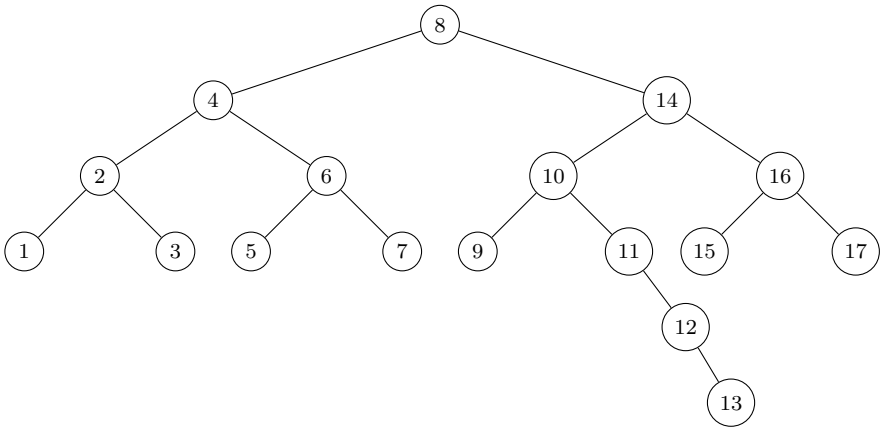
We are left with a binary search tree which has been properly packed into our memory in total time $O(n)$.



(a) Part 1. of ApproxMWPaging



(b) Part 2. of ApproxMWPaging



(c) Part 3. of ApproxMWPaging

Memory Location	Node	Left Child Location	Right Child Location
1	8	2	3
2	4	4	5
3	14	6	7
4	2	8	9
5	6	10	11
6	10	12	13
7	16	14	15
8	1	—	—
9	3	—	—
10	5	—	—
11	7	—	—
12	9	—	—
13	11	—	16
14	15	—	—
15	17	—	—
16	12	—	17
17	13	—	—

(d) Part 4. of ApproxMWPaging

Figure 4.1: An example of the ApproxMWPaging algorithm.

4.5 Expected Cost of ApproxMWPaging

First, we bound the depth of nodes in our BST T. The depth of a key B_i (or (B_{i-1}, B_i) for gaps) is defined as $d_T(B_i)$ (resp. $d_T((B_{i-1}, B_i))$). Note that depth is the number of edges between a node and the root (i.e. the depth of the root is 0). As in the work of Bose

and Douïeb, let $m = \max(n - 3P, P) - 1 \geq \frac{n}{4} - 1$ where P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves (gaps) [15].

Lemma 4.5.1. *For a key B_i,*

$$d_T(B_i) \leq \lg(\frac{1}{p_i}).$$

For a key (B_{i-1}, B_i),

$$d_T((B_{i-1}, B_i)) \leq \lg(\frac{1}{q_i}) + 2$$

for all gaps, and

$$d_T((B_{i-1}, B_i)) \leq \lg(\frac{1}{q_i}) + 1$$

for at least m of them (and the two extremal gaps, (B_0, B_1) and (B_n, B_{n+1})).

Proof. First we note that in the tree T' we build using Bose and Douïeb's multiway tree algorithm, the maximum depth of keys (call this $d_{T'}(B_i), d_{T'}(B_{i-1}, B_i)$) for a page size m_1 is [15]:

$$\begin{aligned} d_{T'}(B_i) &\leq \lfloor log_{m_1}(\frac{1}{p_i}) \rfloor \\ d_{T'}(B_{i-1}, B_i) &\leq \lfloor log_{m_1}(\frac{2}{q_i}) \rfloor + 1 \text{ for all gaps, and} \\ d_{T'}(B_{i-1}, B_i) &\leq \lfloor log_{m_1}(\frac{1}{q_i}) \rfloor + 1 \text{ for at least } m \text{ of them (and the two extremal gaps)} \end{aligned}$$

As explained in the paper, these follow from Lemmas 1 and 2 of Bose and Douïeb [15].

Inside a page, we make a balanced (ignoring weight) BST, so each key has a depth within a page of at most $\lfloor \lg(m_1) \rfloor$. Since our algorithm always connects the root of the BST made for a page to a key in the BST made for the page's parent, a key B_i has a *page depth* (the number of unique pages accessed in order to access the key) of at most the bounds on $d_{T'}(B_i)$ and $d_{T'}(B_{i-1}, B_i)$ described. Since we examine at most $\lceil \lg(m_1) \rceil$ keys

within any one page, (and only 1 gap in a leaf page) a key's depth is at most

$$\begin{aligned} & d_T(B_i) \leq \lfloor \lg(m_1) \rfloor \lfloor log_{m_1}(\frac{1}{p_i}) \rfloor \\ \implies & d_T(B_i) \leq \lg(m_1) \cdot log_{m_1}(\frac{1}{p_i}) \\ \implies & d_T(B_i) \leq \lg(\frac{1}{p_i}). \end{aligned}$$

For an unsuccessful search

$$\begin{aligned} & d_T((B_{i-1}, B_i)) \leq \lfloor \lg(m_1) \rfloor \lfloor log_{m_1}(\frac{2}{q_i}) \rfloor + 1 \\ \implies & d_T((B_{i-1}, B_i)) \leq \lg(m_1) \cdot log_{m_1}(\frac{2}{q_i}) + 1 \\ \implies & d_T((B_{i-1}, B_i)) \leq \lg(\frac{1}{q_i}) + 2 \text{ for all gaps, and} \\ \implies & d_T((B_{i-1}, B_i)) \leq \lg(\frac{1}{q_i}) + 1 \text{ for at least } m \text{ of them (and the two extremal gaps).} \end{aligned}$$

□

Next, we bound the cost of search for each key and each gap. Let $m'_j = \sum_{k \leq j} m_j$. We define $m'_0 = 0$.

Lemma 4.5.2. *For any key B_i, if*

$$\begin{aligned} k = & min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(B_i) \text{ then} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1. \end{aligned}$$

Let $par(B_{i-1}, B_i)$ represent the parent of the node for gap (B_{i-1}, B_i) in T. If $n \geq 1$, then for any gap (B_{i-1}, B_i), if

$$\begin{aligned} k = & min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(par(B_{i-1}, B_i)) \text{ then} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor \text{ and} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor \text{ for at least } m \text{ of them (and the two extremal gaps).} \end{aligned}$$

Proof. Knowing a key's depth, its location in T (since we use a BFS to place nodes into memory) can be bounded as follows:

$$location(B_i) \leq 2^{d_T(B_i)+1} - 1.$$

From Lemma 4.5.1 we know that $d_T(B_i) \leq \lg(\frac{1}{p_i})$. Thus,

$$\begin{aligned} location(B_i) &\leq 2^{\lg(\frac{1}{p_i})+1} - 1 \\ location(B_i) &\leq 2^{\lg(\frac{2}{p_i})} - 1 \\ location(B_i) &\leq \frac{2}{p_i} - 1. \end{aligned}$$

A gap's location can be bounded as follows:

$$location(B_{i-1}, B_i) \leq 2^{d_T((B_{i-1},B_i))+1} - 1.$$

Because we placed our tree T into memory in BFS order, we can also bound the depth of a gap's parent (we assume for the remainder of this proof that the tree is non-trivial, i.e. $n > 0$).

$$\begin{aligned} location(par(B_{i-1}, B_i)) &\leq \lfloor \frac{location(B_{i-1}, B_i)}{2} \rfloor \\ location(par(B_{i-1}, B_i)) &\leq \lfloor \frac{2^{d_T((B_{i-1},B_i))+1} - 1}{2} \rfloor \\ location(par(B_{i-1}, B_i)) &\leq \lfloor 2^{d_T((B_{i-1},B_i))} - \frac{1}{2} \rfloor \end{aligned}$$

Thus, for any gap (B_{i-1}, B_i),

$$\begin{aligned} location(par(B_{i-1}, B_i)) &\leq \lfloor 2^{\lg(\frac{1}{q_i})+2} - \frac{1}{2} \rfloor \text{ using Lemma 4.5.1} \\ location(par(B_{i-1}, B_i)) &\leq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor \text{ for all gaps.} \end{aligned}$$

Moreover, a similar explanation shows that

$$location(par(B_{i-1}, B_i)) \leq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor \text{ for at least } m \text{ (and the two extremal) gaps.}$$

□

Lemma 4.5.3. *The cost of searching for key B_i or gap (B_{i-1}, B_i), ($C(B_i)$ and $C(B_{i-1}, B_i)$ respectively) can be bounded as follows:*

$$C(B_i) \leq \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor \right) \cdot c_{k'} + \left(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor \right) \cdot c_k$$
$$\text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1 \right)$$

$$C(B_{i-1}, B_i) \leq \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor \right) \cdot c_{k'} + \left(lg(\frac{1}{q_i}) + 2 - \lfloor \lg(m'_{k-1} + 1) \rfloor \right) \cdot c_k$$
$$\text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor \right)$$
for all gaps, and

$$C(B_{i-1}, B_i) \leq \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor \right) \cdot c_{k'} + \left(lg(\frac{1}{q_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor \right) \cdot c_k$$
$$\text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor \right)$$
for at least m (and the two extremal) gaps.

Proof. Consider accessing each key along the path from the root to key B_i. We will examine $d_T(B_i)$ keys. We access one key at depth 0, one key at depth 1, and so on. Because the tree is packed into memory in BFS order, a key a depth j will be at memory index at most $2^j - 1$. Now, consider how many levels of the binary search tree T will fit in m_1. In order for all keys of depth j (and higher) to be in m_1 we need:

$$2^{j+1} - 1 \leq m_1 \implies j \leq \lg(m_1 + 1) - 1.$$

Thus, at least $\lfloor \lg(m_1 + 1) \rfloor$ levels of T (since the root node has depth 0) will completely fit on m_1. We expand this concept for arbitrary memory level m_k. The last level of T to completely fit on m_k or higher memories is the maximum s such that:

$$2^s - 1 \leq m'_k \implies s \leq \lg(m'_k + 1).$$

Thus, at least $\lfloor \lg(m'_j + 1) \rfloor$ levels of T fit on m_j or higher levels of memory. On our search for B_i, we make at least $\lfloor \lg(m_1 + 1) \rfloor$ checks for elements located at memory m_1. This costs a total of

$$\lfloor \lg(m_1 + 1) \rfloor \cdot c_1.$$

Let k be the minimum memory level such that $m'_k \geq location(B_i)$. For each memory level m'_k for $0 < k' < k$, we make at least $\lfloor \lg(m'_{k'} + 1) \rfloor$ checks in $m'_{k'}$ or higher memories. Of these checks, at least $\lfloor \lg(m'_{k'-1} + 1) \rfloor$ are in memory levels strictly higher up in the memory hierarchy than k'. Since $c_\alpha > c_\beta$ for $\alpha > \beta$, an upper bound on the cost of searching for all elements in memory level $0 < k' < k$ on the path from the root to $B(i)$ is:

$$(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor) \cdot c'_k.$$

Finally, we can upper bound the cost of search within m_k by assuming that all remaining searches take place at memory level k. The searches at level k will cost at most:

$$(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor) \cdot c_k.$$

Combining the above three equations (and summing over every memory level) gives us the total cost of searching for a key in the deepest part of the tree:

$$\begin{aligned} C(B_i) \leq & \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor \right) \cdot c_{k'} + \\ & \left(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor \right) \cdot c_k \\ & \text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(B_i) \right) \end{aligned}$$

Similarly, to search for a gap (B_{i-1}, B_i) we must access each of its descendants in turn (but not the leaf node representing that gap itself). Our search effectively ends at the parent of (B_{i-1}, B_i). Thus,

$$C(B_{i-1}, B_i) \leq \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_{k'} + \left(lg(\frac{1}{q_i}) + 2 - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_k \text{ such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(par(B_{i-1}, B_i))\right)$$

For all gaps, and,

$$C(B_{i-1}, B_i) \leq \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_{k'} + \left(lg(\frac{1}{q_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_k \text{ such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(par(B_{i-1}, B_i))\right)$$

for at least m (and the two most extremal) gaps.

Plugging in our upper bounds on $location(B_i)$ and $location(par(B_{i-1}, B_i))$ from Lemma 4.5.2 immediately gives us the desired result.

□

In order to use Lemma 4.5.3, we define a function C', which consumes a key B_i as follows:

$$C'(B_i) := \sum_{k'=1}^{k-1} \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor\right) \cdot c_{k'} + \left(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_k \text{ such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1\right)$$

We also define a function C'', which consumes a gap (B_{i-1}, B_i). First, we define:

$$Q = \{q_i : q_i \text{ among smallest } m \text{ gaps excluding the extremal gaps}\} \cup \{q_0, q_n\}$$

$$C''((B_{i-1}, B_i)) := \begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor\right) \cdot c_{k'} + \\ & \left(lg(\frac{1}{q_i}) + 2 - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_k \\ & \text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor\right) \end{cases} \quad \text{if } q_i \notin Q$$

$$\begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'} + 1) \rfloor - \lfloor \lg(m'_{k'-1} + 1) \rfloor\right) \cdot c_{k'} + \\ & \left(lg(\frac{1}{q_i}) + 1 - \lfloor \lg(m'_{k-1} + 1) \rfloor\right) \cdot c_k \\ & \text{such that } \left(k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor\right) \end{cases} \quad \text{otherwise}$$

We can now bound the expected cost of search, C, in T explicitly.

Theorem 4.5.4.

$$C \leq \sum_{i=1}^{n} p_i \cdot C'(B_i) + \sum_{i=1}^{n+1} q_i \cdot C''((B_{i-1}, B_i))$$

Proof. The total expected cost of search is simply the sum of the weighted cost of search for all keys and gaps multiplied by their respective probabilities.

$$C \leq \sum_{i=1}^{n} p_i \cdot C(B_i) + \sum_{j=1}^{n+1} q_j \cdot C(B_{i-1}, B_i)$$

We note that functions C' and C'' are exactly equal to the upper bounds of cost of search for a key or gap respectively as defined in Lemma 4.5.3. Hence, we can simply plug them into the equation above and get an upper bound on the expected cost of search in T:

$$C \leq \sum_{i=1}^{n} p_i \cdot C'(B_i) + \sum_{j=1}^{n+1} q_j \cdot C''((B_{i-1}, B_i))$$

□

In order to get a more cleaner (albeit weaker) bound in terms of the entropy of the distribution, we now describe W, the cost of searching for a key located at the deepest node of tree T. As defined in Thite's work, we let h be the smallest j such that $m'_j \geq n$. Let $D(T)$ be the height of T (the depth of the deepest node in the tree).

Lemma 4.5.5.

$$W \leq \sum_{k=1}^{h-1} \left(\lfloor \lg(m'_k + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1 \rfloor \right) \cdot c_k + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

Proof. We can follow the exact same logic as in Lemma 4.5.3 with the depth of our node as $D(T)$, and its memory level h. This immediately gives us the desired result.

□

This leads us to the following upper bound for W.

Lemma 4.5.6. *Assuming that $l \neq 1$:*

$$W < D(T) \cdot c_h$$

Proof. We can rearrange Lemma 4.5.5 as follows:

$$W \leq D(T) \cdot c_h - \sum_{k=1}^{h-1} \lfloor \lg(m'_k + 1) \rfloor \cdot (c_{k+1} - c_k)$$

Since we have that $c_k > c_{k-1}$ for all k, $\sum_{k=1}^{h-1} \lfloor \lg(m'_k + 1) \rfloor \cdot (c_{k+1} - c_k)$ is strictly positive. This instantly gives the result desired. □

Note that $\frac{W}{D(T)}$ represents the average cost per memory access when accessing the deepest (and most costly) element of our tree. Since our costs of access are monotonically increasing as we move deeper in the tree, we will see that $\frac{W}{D(T)}$ can be used as an upper bound for the average cost per memory access when searching for any element of T.

Lemma 4.5.7. *The cost of searching for a keys B_i and (B_{i-1}, B_i) ($C(B_i)$ and $C((B_{i-1}, B_i))$ respectively) can be bounded as follows:*

$$C(B_i) \leq (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{p_i}) + 1) \cdot c_h$$

$$C((B_{i-1}, B_i)) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{q_i}) + 2) \cdot c_h \text{ for all gaps, and}$$

$$C((B_{i-1}, B_i)) \leq (\lg(\frac{1}{q_i}) + 1) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{q_i}) + 1) \cdot c_h \text{ for at least } m \text{ (and the two extremal) gaps.}$$

Proof. From Lemma 4.5.1 we have a bound on the depth of keys B_i and (B_{i-1}, B_i). We must do $d_T(B_i) + 1$ accesses to find a key and $d_T((B_{i-1}, B_i))$ accesses to find a gap. Note that since our tree is stored in BFS order in memory, whenever we examine a key's child, it will be at a memory location of at least the same, if not higher cost (by being in the same or a deeper page). Thus, the cost of accessing the entire path from the root to a specific key can be upper bounded by multiplying the length of this path by the average cost per memory access of the most expensive (and deepest) key of the tree (this is exactly $\frac{W}{D(T)}$). Note that when searching for keys, we must search along the entire path from root to the key in question, while we need only examine the path from the root to the parent of a key for unsuccessful (B_{i-1}, B_i) searches. Combining Lemmas 4.5.1, 4.5.5 and 4.5.6 gives

$$\begin{aligned} & C(B_i) \leq (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} \\ \implies & C(B_i) < (\lg(\frac{1}{p_i}) + 1) \cdot \frac{D(T) \cdot c_h}{D(T)} \\ \implies & C(B_i) < (\lg(\frac{1}{p_i}) + 1) \cdot c_h \end{aligned}$$

For gaps we have that

$$\begin{aligned} & C((B_{i-1}, B_i)) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{W}{D(T)} \\ \implies & C((B_{i-1}, B_i)) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{D(T) \cdot c_h}{D(T)} \\ \implies & C((B_{i-1}, B_i)) < (\lg(\frac{1}{q_i}) + 2) \cdot c_h \text{ for all gaps, and} \\ \implies & C((B_{i-1}, B_i)) < (\lg(\frac{1}{q_i}) + 1) \cdot c_h \text{ for at least } m \text{ (and the two extremal) gaps.} \end{aligned}$$

□

We can now bound the expected cost of search using the bounds for the cost of each key.

Theorem 4.5.8.

$$C \leq \frac{W}{D(T)} \cdot \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{rank[i]} \right) \text{ and}$$

$$C < \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{rank[i]} \right) \cdot c_h$$

where $q_{rank[i]}$ is the i^{th} smallest access probability among gaps except q_0 and q_n.

Proof. As in Lemma 4.5.4, the total expected cost of search is simply the sum of the weighted cost of search for all keys multiplied by the probability of searching for each key. Given our last lemma, we have that:

$$C \leq \sum_{i=1}^{n} p_i \cdot C(B_i) + \sum_{j=1}^{n+1} q_j \cdot C((B_{i-1}, B_i))$$

$$\implies C \leq \sum_{i=1}^{n} p_i \cdot (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} + \left(\sum_{i=0}^{n} q_i(\lg(\frac{1}{q_i}) + 2) - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right) \cdot \frac{W}{D(T)}$$

$$\implies C \leq \frac{W}{D(T)} \left(\sum_{i=1}^{n} p_i \lg(\frac{1}{p_i}) + \sum_{i=0}^{n} q_i \lg(\frac{1}{q_i}) + \sum_{i=1}^{n} p_i + 2\sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right)$$

$$\implies C \leq \frac{W}{D(T)} \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right)$$

By Lemma 4.5.6 this gives:

$$C < \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right) \cdot c_h$$

□

4.6 Approximate Binary Search Trees of De Prisco and De Santis with Extensions by Bose and Douïeb

We provide another approach to building the approximately optimal BST under the HMM model. This approach uses the approximate BST solution (in the simple RAM model) of De Prisco and De Santis [18] (modified by Bose and Douïeb [15]). We explain the method here.

As in the classic Knuth problem, we are given a set of n probabilities of searching for keys $(p_1, p_2, ..., p_n)$, as well as $n+1$ probabilities of unsuccessful searches $(q_0, q_1, ..., q_n)$. De Prisco and De Santos give an algorithm which constructs a binary search tree in $O(n)$ time with an expected cost of at most [18]

$$H + 1 - q_0 - q_n + q_{max}$$

where q_{max} is the maximum probability of an unsuccessful search. This was later modified by Bose and Douïeb (the same paper described in section 4.3) to have an improved bound of [15]

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}.$$

Here, P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the gaps and $m' = \max(2n - 3P, P) - 1 \geq \frac{n}{2} - 1$. Moreover, $pq_{\text{rank}[i]}$ is the i^{th} smallest access probability among all keys and gaps except q_0 and q_n.

First, we a high level explanation of the algorithm of De Prisco and De Santis and then explain the extensions of Bose and Douïeb. De Prisco and De Santis' algorithm occurs in three phases.

Phase 1 An auxiliary probability distribution is created using $2n$ zero probabilities, along with the $2n + 1$ successful and unsuccessful search probabilities. Yeung's linear time alphabetic search tree algorithm is used with the $2n + 1$ successful and unsuccessful search probabilities used as gaps of the new tree created [59]. This is referred to as the *starting tree*.

Phase 2 What's known as the *redundant tree* is created by moving p_i keys up the *starting tree* to the lowest common ancestor of keys q_{i-1} and q_i. The keys which used to be called p_i are relabelled to $old.p_i$.

Phase 3 The *derived tree* is constructed from the *redundant tree* by removing redundant edges. Edges to and from nodes which represented zero probability keys are deleted. This *derived tree* is a binary search tree with the expected search cost described.

In Bose and Douïeb's work, they explain how they can substitute their algorithm for Yeung's linear time alphabetic search tree algorithm which results in a better bound (as described above). We use the updated version (by Bose and Douïeb) of De Prisco and De Santis' algorithm as a subroutine in the sections to follow.

Let T_1 be the *starting tree* created after **Phase 1** of the algorithm using Bose and Douïeb's method. As explained Section 4 of their work [15], every leaf (both keys and gaps from the original dataset) with probability α have depth at most:

$$d_{T_1}(\alpha) \leq \begin{cases} \lfloor \lg \frac{1}{\alpha} \rfloor + 1 \text{ for at least } \max\{2n - 3P\} - 1 \text{ and the extremal gaps of } T_1 \\ \lfloor \lg \frac{1}{\alpha} \rfloor + 2 \text{ otherwise} \end{cases}$$

Moreover, each key from our original data set has its depth decreased by at least 2 while each leaf from our original data set, except for possibly one, has its depth reduced by at least 1 in the next two phases of the algorithm. We wish to write this out as an explicit Lemma for use in section 4.8. First, we define R:

$$R = \{\alpha : \alpha \text{ among the } m' \text{ smallest access probabilities for keys or gaps excluding the extremal gaps}\} \cup \{q_0, q_n\} - \{q_{max}\}$$

Lemma 4.6.1. *Let T be a tree made using the approximate binary search tree algorithm of De Prisco and De Santis with extensions by Bose and Douïeb. For a key B_i,*

$$d_T(B_i) \leq \begin{cases} \lfloor \lg(\frac{1}{p_i}) \rfloor - 1 & \text{if } p_i \in R \\ \lfloor \lg(\frac{1}{p_i}) \rfloor & \text{otherwise.} \end{cases}$$

For a gap (B_{i-1}, B_i),

$$d_T((B_{i-1}, B_i)) \leq \begin{cases} \lfloor \lg(\frac{1}{q_i}) \rfloor & \text{if } q_i \in R \\ \lfloor \lg(\frac{1}{q_i}) \rfloor + 2 & \text{if } q_i \text{ is } q_{max} \\ \lfloor \lg(\frac{1}{p_i}) \rfloor + 1 & \text{otherwise.} \end{cases}$$

Proof. This follows directly from the explanation above. □

4.7 Algorithm ApproxBSTPaging

Our second solution to create an approximately optimal BST under the HMM model works as follows:

1. First, we create a BST T using the algorithm of De Prisco and De Santis [18] (as updated by Bose and Douïeb [15]). This takes $O(n)$ time.

2. In a similar fashion to step 4) of section 4.4, we pack keys from T into memory in a breadth-first search order starting from the root. This relatively simple traversal also takes $O(n)$ time.

We are left with a binary search tree which is properly packed into memory in total time $O(n)$.

4.8 Expected Cost of ApproxBSTPaging

The explanation here follows in a similar manner to that of section 4.5. First, we use Lemma 4.6.1 to bound the depth of a node in the memory hierarchy given its probability. The proof follows in a similar fashion to Lemma 4.5.2. Recall that $m'_j = \sum_{k \leq j} m_j$ and $m'_0 = 0$. Moreover, recall R:

$$R = \{\alpha : \alpha \text{ among the } m' \text{ smallest access probabilities for keys or gaps excluding the extremal gaps}\} \cup \{q_0, q_n\} - \{q_{max}\}$$

Lemma 4.8.1. *For any key B_i, if*

$$\begin{aligned} k = & min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(B_i) \text{ then} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \frac{1}{p_i} - 1 \text{ if } p_i \in R \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1 \text{ otherwise.} \end{aligned}$$

Let $par(B_{i-1}, B_i)$ represent the parent of the node for gap (B_{i-1}, B_i) in T. If $n \geq 1$, then for any gap (B_{i-1}, B_i), if

$$\begin{aligned} k = & min_{j \in \{1,\ldots,h\}} \mid m'_j \geq location(par(B_{i-1}, B_i)) \text{ then} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{1}{q_i} - \frac{1}{2} \rfloor \text{ if } q_i \in R \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor \text{ else if } q_i = q_{max} \\ & k = min_{j \in \{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor \text{ otherwise.} \end{aligned}$$

Proof. As before, knowing a key's depth, its location in T can be bounded as follows:

$$location(B_i) \leq 2^{d_T(B_i)+1} - 1.$$

Hence, using Lemma 4.6.1 we get:

$$location(B_i) \leq 2^{\lfloor \lg(\frac{1}{p_i}) \rfloor - 1 + 1} - 1$$
$$location(B_i) \leq 2^{\lg(\frac{1}{p_i})} - 1$$
$$location(B_i) \leq \frac{1}{p_i} - 1 \text{ when } p_i \in R \text{ and}$$

$$location(B_i) \leq 2^{\lfloor \lg(\frac{1}{p_i}) \rfloor + 1} - 1$$
$$location(B_i) \leq 2^{\lg(\frac{2}{p_i})} - 1$$
$$location(B_i) \leq \frac{2}{p_i} - 1 \text{ otherwise.}$$

Recall a gap's location and its parent's location can be bounded as follows:

$$location(B_{i-1}, B_i) \leq 2^{d_T((B_{i-1}, B_i))+1} - 1$$
$$location(par(B_{i-1}, B_i)) \leq \lfloor 2^{d_T((B_{i-1}, B_i))} - \frac{1}{2} \rfloor$$

Thus, for any gap (B_{i-1}, B_i),

$$location(par(B_{i-1}, B_i)) \leq \lfloor 2^{\lg(\frac{1}{q_i})} - \frac{1}{2} \rfloor$$
$$location(par(B_{i-1}, B_i)) \leq \lfloor \frac{1}{q_i} - \frac{1}{2} \rfloor \qquad \text{if } q_i \in R$$

$$location(par(B_{i-1}, B_i)) \leq \lfloor 2^{\lg(\frac{1}{q_i})+2} - \frac{1}{2} \rfloor$$
$$location(par(B_{i-1}, B_i)) \leq \lfloor \frac{4}{q_i} - \frac{1}{2} \rfloor \qquad \text{if } q_i = q_{max}$$

$$location(par(B_{i-1}, B_i)) \leq \lfloor 2^{\lg(\frac{1}{q_i})+1} - \frac{1}{2} \rfloor$$
$$location(par(B_{i-1}, B_i)) \leq \lfloor \frac{2}{q_i} - \frac{1}{2} \rfloor \qquad \text{otherwise.}$$

□

As in section 4.5 we can now bound the cost of search for keys and gaps.

Lemma 4.8.2. *The cost of searching for key B_i or gap (B_{i-1}, B_i), ($C(B_i)$ and $C(B_{i-1}, B_i)$ respectively) using the ApproxBSTPaging algorithm can be bounded as follows:*

$$C(B_i) \leq \begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{p_i}) - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \textit{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \frac{1}{p_i} - 1 \textit{ if } p_i \in R \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \textit{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1 \textit{ otherwise.} \end{cases}$$

$$C((B_{i-1}, B_i)) \leq \begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \textit{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{1}{q_i} - \frac{1}{2}\rfloor \textit{ if } q_i \in R \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) + 2 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \textit{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2}\rfloor \textit{ else if } q_i = q_{max} \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) + 1 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \textit{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2}\rfloor \textit{ otherwise.} \end{cases}$$

Proof. The proof follows an almost identical set of logic to Lemma 4.5.3 in terms of how far into the tree we search for each node. We then can simply plug in the maximum depth of nodes from Lemma 4.6.1, and the maximum location in the memory hierarchy from Lemma 4.8.1 and arrive at the result. □

In order to use Lemma 4.8.2, we define a function C', which consumes a key B_i as

follows:

$$C'(B_i) := \begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{p_i}) - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \text{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \frac{1}{p_i} - 1 \text{ if } p_i \in R \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{p_i}) + 1 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \text{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \frac{2}{p_i} - 1 \text{ otherwise.} \end{cases}$$

We also define a function C'', which consumes a gap (B_{i-1}, B_i):

$$C''((B_{i-1}, B_i)) := \begin{cases} \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \text{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{1}{q_i} - \frac{1}{2}\rfloor \text{ if } q_i \in R \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) + 2 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \text{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{4}{q_i} - \frac{1}{2}\rfloor \text{ else if } q_i = q_{max} \\ \sum_{k'=1}^{k-1} & \left(\lfloor \lg(m'_{k'}+1)\rfloor - \lfloor \lg(m'_{k'-1}+1)\rfloor\right)\cdot c_{k'} + \\ & \left(\lg(\frac{1}{q_i}) + 1 - \lfloor \lg(m'_{k-1}+1)\rfloor\right)\cdot c_k \\ & \text{such that } k = min_{j\in\{1,\ldots,h\}} \mid m'_j \geq \lfloor \frac{2}{q_i} - \frac{1}{2}\rfloor \text{ otherwise.} \end{cases}$$

We can now bound the expected cost of search, C, in T explicitly.

Theorem 4.8.3.

$$C \leq \sum_{i=1}^{n} p_i \cdot C'(B_i) + \sum_{i=1}^{n+1} q_i \cdot C''((B_{i-1}, B_i))$$

Proof. The total expected cost of search is simply the sum of the weighted cost of search for all keys and gaps multiplied by their respective probabilities.

$$C \leq \sum_{i=1}^{n} p_i \cdot C(B_i) + \sum_{j=1}^{n+1} q_j \cdot C(B_{i-1}, B_i)$$

As in Theorem 4.5.4, C' and C'' are exactly equal to the upper bounds of cost of search for a key or gap respectively as defined in Lemma 4.8.2. We simply plug them into the equation above and get an upper bound on the expected cost of search in T:

$$C \leq \sum_{i=1}^{n} p_i \cdot C'(B_i) + \sum_{j=1}^{n+1} q_j \cdot C''((B_{i-1}, B_i))$$

□

Like in section 4.5, we wish to make a cleaner albeit weaker bound in terms of the entropy of the probability distribution. As explained in the Bose and Douïeb paper, the average path length search cost of the tree created by their algorithm is at most: [15]

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} p q_{\text{rank}[i]}$$

We call this value P_T (the average search cost of tree T). Similar to the proof in section 4.5, if we can bound the cost of search for a given path length, then we can form a bound on the average cost of search in the HMM model. As before, we can describe the cost of searching for a key located at deepest node of tree T: W. Recall, $m'_j = \sum_{k \leq j} m_k$, $m'_0 = 0$ and let h be the smallest j such that $m'_j \geq n$.

Lemma 4.8.4. *When using the ApproxBSTPaging, the cost of searching for a node deepest in the tree is at most:*

$$W \leq \sum_{k=1}^{h-1} \left(\lfloor \lg(m'_k + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1) \rfloor \right) \cdot c_k + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

Proof. Since we are simply putting keys into memory in BFS order, and all we use is the height of the tree and the memory hierarchy, the proof is identical to that of Lemma 4.5.5. □

Since we have the same result as Lemma 4.5.5, this immediately implies Lemma 4.5.6 is true as well. Assuming that $l \neq 1$, we have that $W < D(T) \cdot c_h$. As in Theorem 4.5.4, $\frac{W}{D(T)}$ represents the average cost per memory access when accessing the deepest (and most costly) element of our tree. Thus, $\frac{W}{D(T)}$ upper bounds the average cost per memory access when searching for any element of T.

Theorem 4.8.5.

$$C \leq (\frac{W}{D(T)}) \cdot (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \text{ and}$$

$$C < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \cdot c_h$$

Proof. Bose and Douïeb show that after using their algorithm for Phase 1 of De Prisco and De Santis algorithm algorithm, every leaf of the *starting tree* (all keys representing successful searches and gaps representing unsuccessful searhces) are at depth at most $\lfloor \lg(\frac{1}{p}) \rfloor + 1$ for at least $\max(2n - 3P, P) - 1$ of $p \in (\{p_1, p_2, ..., p_n\} \cup \{q_0, q_1, ..., q_n\})$ and $\lfloor \lg(\frac{1}{p}) \rfloor + 2$ for all others. Recall that P is the number of peaks in the probability distribution $q_0, p_1, q_1, ..., p_n, q_n$. After phases 2 and 3 of the algorithm, each key has its depth decrease by 2, and all gaps (except one) move up the tree by one.

$$C \leq \sum_{i=1}^{n} \left(p_i \cdot \frac{depth(p_i) + 1}{D(T)} \cdot W \right) + \sum_{i=0}^{n} \left(q_i \cdot \frac{(depth(q_i))}{D(T)} \cdot W \right)$$

$$\implies C \leq \frac{W}{D(T)} \left(\sum_{i=1}^{n} (p_i \cdot (depth(p_i) + 1)) + \sum_{i=0}^{n} (q_i \cdot (depth(q_i))) \right)$$

$$\implies C \leq \frac{W}{D(T)} (\text{WeightedAveragePathLength}(T))$$

$$\implies C \leq \left(\frac{W}{D(T)} \right) \cdot \left(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]} \right) \text{ and by Lemma 4.5.6}$$

$$\implies C < \left(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]} \right) \cdot c_h$$

□

4.9 Improvements over Thite in the HMM_2 Model

The HMM_2 model is the same as the general HMM model with the added constraint that there are only two types of memory (slow and fast). In Thite's thesis, he proposed both an optimal solution to the problem, as well as an approximate solution (Algorithm Approx-BST) that runs in time $O(n \lg(n))$ [52]. we first show that:

Lemma 4.9.1. *The quality of approximation of Thite's algorithm has a bound of:*

$$c_2(H + 1 + \sum_{i=0}^{n} q_i)$$

Proof. Specifically, in Lemma 14 in 3.4.2.3 "Quality of approximation" in Thite's thesis, he proves that $\delta(z_k) = l + 2$. Here, $\delta(z_k)$ represents the depth of a leaf node z_k. Note that Thite considers the depth of the root to be 1 instead of 0 which updates how the cost of search is calculated accordingly. l represents the depth of recursion of the *Approx-BST* algorithm. In Lemma 15, Thite goes on to prove that $q_k \leq 2^{-\delta(z_k)+2}$. In this proof, Thite shows that $q_k \leq 2^{-l+1}$, but makes a mistake when substituting in $l = \delta(z_k) - 2$ and gets $q_k \leq 2^{-\delta(z_k)+2}$ while the correct bound is $q_k \leq 2^{-\delta(z_k)+3}$. This updated bound would change his depth bound in Lemma 16 from $\delta(z_k) \leq \lfloor \lg(\frac{1}{q_k}) \rfloor + 2$ to $\delta(z_k) \leq \lfloor \lg(\frac{1}{q_k}) \rfloor + 3$. Finally, substituting into his final equation for the upper bound on the expected cost of search for the tree would give:

$$\begin{aligned}
&\sum_{i=1}^{n} \left(c_2 p_i \delta(B_i) + \sum_{i=0}^{n} c_2 q_i (\delta(q_i) - 1) \right) \\
\leq &\sum_{i=1}^{n} \left(c_2 p_i (\lg(\frac{1}{p_i}) + 1) + \sum_{i=0}^{n} c_2 q_i (\lg(\frac{1}{q_i}) + 3 - 1) \right) \\
\leq &c_2 \cdot \left(H + 1 + \sum_{i=0}^{n} q_i \right)
\end{aligned}$$

□

This is of particular interest because if Thite's bound on *Algorithm Approx-BST* had been correct, then in the case where $c_2 = c_1$ (typical RAM model), Thite's method would have provided a strict improvement over the work of Bose and Douïeb [15] which seems

unlikely since Thite used the BST approximation algorithm of Mehlhorn from 1984 [47] (much before the work of Bose and Douïeb).

By simply substituting in for $l = 2$ we immediately get that, under this HMM_2 model, both ApproxMWPaging and ApproxBSTPaging provide strict improvements over Thite's *Algorithm Approx-BST*.

Theorem 4.9.2. *In the HMM_2 model, ApproxMWPaging has an expected cost of at most*

$$C_{ApproxMWPaging} < (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{rank[i]}) \cdot c_2$$

and ApproxBSTPaging has an expected cost of at most

$$C_{ApproxBSTPaging} < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \cdot c_2.$$

ApproxMWPaging and ApproxBSTPaging do so in $O(n)$ time.

Proof. We can directly sub $l = 2$ into Theorems 4.5.8 and 4.8.5 to get the desired result (the running times are as explained in sections 4.4 and 4.7). □

Since both ApproxMWPaging and ApproxBSTPaging run in time $o(n \lg(n))$ (the time of Thite's *Approx-BST* algorithm) and we can see that:

$$\begin{aligned} C_{ApproxMWPaging} &< (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}) \cdot c_2 \\ &< c_2(H + 1 + \sum_{i=0}^{n} q_i) \\ C_{ApproxBSTPaging} &< (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}) \cdot c_2 \\ &< c_2(H + 1 + \sum_{i=0}^{n} q_i) \end{aligned}$$

Both methods provide a strictly better approximation and run faster than the *Algorithm Approx-BST* of Thite.

Chapter 5

Binary Trees On Unordered Sequences of Probabilities

In this chapter we examine a problem related to the initial optimal binary search tree problem of Knuth [43]. The n keys (represented using simply their probabilities) provided will no longer have an ordering and can be rearranged as we please before constructing our binary tree. We also do not have any restrictions on which keys can be internal nodes and which keys can be leaves. This problem is very similar to constructing optimal prefix-free binary codes which is solved using a Huffman coding. The problem differs however because we must place the n probabilities at internal *and* leaf node positions and cannot add extra nodes. Moreover, when searching, our cost model charges 1 for each internal node we examine but does not charge for leaf nodes (since we do not need to 'examine' them). Under this cost model, we show that our $O(n \lg(n))$ time algorithm GREEDY-MS is within an additive constant of $\frac{n+1}{2n}$ from optimal.

5.1 The Binary Tree On Unordered Sequences of Probabilities Problem

Consider a multiset (a set with possible duplicate values) of n probabilities: $M = \{p_1, p_2, ..., p_n\}$ such that $\sum_{i=1}^{n} p_i = 1$. We assume n is odd. Our goal is to create a binary tree T which minimizes the expected search cost, C_T^S, of nodes. We call this cost model the the **S**tandard

Model since it is standard among other optimal BST problems.

$$C_T^S = \sum_{i=1}^{n} p_i(b_i + 1) - \sum_{p_i \in L_T} p_i \tag{5.1}$$

Here, b_i is the *depth* of the i'th key of M and L_T is the set of leaves of T. We subtract the weight of the leaves of the tree since we need one less comparison to return a pointer a leaf node (as in the original optimal BST problem). Note that this is analogous to our previous cost model (equation 1.1) but with a simple modification of how we describe the set of leaves (L_T instead of $\{q_i\}$ for appropriate i).

5.2 The GREEDY-MS Algorithm is Within $\frac{n+1}{2n}$ of Optimal

First, we consider a new cost model for this problem, the ***E**xpected Path Length Model*, which has cost C_T^E defined as follows:

$$C_T^E = \sum_{i=1}^{n} p_i(b_i + 1) \tag{5.2}$$

The model treats leaves and internal nodes the same, charging 1 for each node that must be examined when searching through the tree from the root for a specific node. This model corresponds to a problem instance under the Standard Model, but with the addition of $n+1$ probabilities with value 0 (which should obviously be placed at the $n+1$ leaf locations).

As described by Golin et al. [28] there is a simple greedy $O(n \lg n)$ time algorithm which solves the problem optimally. We call this algorithm GREEDY-MS:

1. First, we create a vector R which is equal to the sorted (from largest to smallest) multiset M.

2. We create a BST T as follows. The root of our tree will be $R[0]$, its two children will be $R[1]$ and $R[2]$, and so on. Formally, $R[i]$ will be placed at location $i+1$ in the BFS order of T.

For completeness we formally prove its optimality under the Expected Path Length Model.

Lemma 5.2.1. *The tree T created by GREEDY-MS for the multiset of probabilities M solves the problem under the Expected Path Length Model optimally.*

Proof. Let T' be an optimal tree under the *Expected Path Length Model*.

Claim 1. For a given depth d, the probabilities of all nodes on level $d+1$ are greater than or equal to the probabilities of all nodes on level d.

Proof. If Claim 1 were not true, suppose p_i and p_j were the two probabilities which contradicted the statement. Swap their positions. The cost of the resulting tree is strictly less, which contradicts the optimality of T'. □

Claim 2. For a given depth d strictly less than the height of T', T' must be full.

Proof. If Claim 2 were not true consider d'; the smallest depth where it is not true. Take a leaf node with depth strictly greater than d' and place it at depth d in T'. The resulting tree costs strictly less than T', another contradiction. □

Taking claims 1 and 2 together means that all optimal trees are full (except at the greatest depth) and always have higher probabilities above lower probabilities. All such trees are identical to T created by GREEDY-MS up to permutations among probabilities at a given depth d, and amongst identical probabilities, but still have the same cost. This completes the proof.

□

In order to prove that GREEDY-MS is within $\frac{n+1}{2n}$ of optimal under the Standard Model we first we must prove a lemma for optimal trees under the Standard Model. We show that for an optimal tree where, for each parent child relationship, the parent has probability at least as large as the child, each leaf node (except one, the minimum) has a unique corresponding internal node with probability at least as big.

We say a leaf p_i (we are using p_i to represent a probability and a node) is *covered* by a unique internal node p'_i if p'_i is an internal node, p_i is a leaf node and $p'_i \geq p_i$.

Lemma 5.2.2. *Let T be a tree such that, for all child parent pairs, the probability of the parent is greater than or equal to that of the child. Let l_{min} be the smallest leaf node by probability. Then*

$$\forall_{p_i \in L_T - \{l_{min}\}} \exists \text{ unique } p_i' \notin \{L_T \cup \{l_{min}\}\} \text{ such that } p_i' \text{ covers } p_i$$

Proof. The proof follows by induction on the height of T. When T has height 1, it trivially holds since we only have a single leaf, which does not need a an internal node to cover it since it is l_{min}.

Suppose for all T (as in the lemma description) with heights strictly less than l, the claim holds. Consider any T (as in the lemma description) with height equal to l. One of two cases arise:

1. If our tree root has two leaf children, then the probability of the root must be at least $\frac{1}{3}$ (since it must be bigger than its children) and $n = 3$. The root node is an internal node which we choose to cover its larger child. The remaining leaf is minimum so we get the desired result.

2. If our tree root has at least one non-leaf child, the by our induction hypothesis, our non-leaf subtrees have internal nodes to cover all but their smallest leaves. Since our root node must have greater probability than any other node in the tree (otherwise our parent child relationship assumption will be contradicted), the root node can *cover* the larger of these two smallest leaves. We maintain all other *covers* from solution to the subtrees with heights at most $l - 1$ (solutions must exist from our induction hypothesis). We get a covering as required.

Combining the above two cases, we get that there exists a covering of all leaf nodes (except the smallest leaf node) by internal nodes by induction. This gives the desired result, completing the proof.

□

Lemma 5.2.3. *Let T be a tree such that, for all child parent pairs, the probability of the parent is greater than or equal to that of the child. Then:*

$$\sum_{p_i \in L_T} p_i \leq \frac{n+1}{2n}$$

Proof. By Lemma 5.2.2 we know that the sum of the probabilities of all internal nodes is at least as large as the sum of the probabilities of all leaf nodes minus the probability of the smallest leaf node. We also know the sum of all probabilities except the smallest leaf node is $1 - l_{min}$. Thus,

$$\sum_{p_i \in L_T} p_i \leq \frac{1}{2} \cdot (1 - l_{min}) + l_{min}$$

Note that this is maximized when l_{min} is maximized. The maximum value for l_{min} is exactly $\frac{1}{n}$, otherwise we contradict our parent-child relationship assumption. Thus,

$$\sum_{p_i \in L_T} p_i \leq \frac{1}{2} \cdot (1 - \frac{1}{n}) + \frac{1}{n}$$

$$\sum_{p_i \in L_T} p_i \leq \frac{n+1}{2n} \text{ as required.}$$

□

We are now ready to prove our main theorem.

Theorem 5.2.4. *The GREEDY-MS Algorithm is within $\frac{n+1}{2n}$ of optimal under the Standard Model for cost.*

Proof. Let T be our tree created by GREEDY-MS. Let R be any tree such that, for all child parent pairs, the probability of the parent is greater than or equal to that of the child. Note that given any tree we can performs a series of parent-child swaps to get such a tree with this property. Moreover this new tree will have cost less than or equal to the cost of the original tree in both the Standard and Expected Path Length models. We can show:

$$C_T^S \leq C_T^E \leq C_R^E = C_R^S + \sum_{p_i \in L_R} p_i \leq C_R^S + \frac{n+1}{2n}.$$

The first inequality comes from the definition of the two models, the second from the optimality of GREEDY-MS under the Expected Path Length model, the equality comes from the definitions of the two models, and the final inequality comes from Lemma 5.2.3. As we noted above, any tree which does not follow the child parent property described is at least as bad, if not worse, under the Standard Model. Thus, T is within $\frac{n+1}{2n}$ of the cost of any tree under the Standard model, completing the proof.

□

Chapter 6

Conclusion and Open Problems

6.1 Conclusion

In this work, we examined several problems related to the optimum BST problem originally proposed (and solved) by Knuth in 1971 [43]. In Chapter 3 we showed that the Modified Entropy Rule first proposed by Güttler, Mehlhorn and Schneider in 1980 had a worst case expected cost of $H + 4$. This improved upon the previous best bound of $c \cdot H + 2$ where $c \approx 1.08$ [32].

In Chapter 4 we examined the problem under a model for external memory. We showed that under the Hierarchical Memory Model (HMM) by Aggarwal et al. our two algorithms ApproxMWPaging and ApproxBSTPaging solved the problem approximately in time $O(n)$ [2]. Moreover in sections 4.5 and 4.8, we showed the two solutions had worst case expected costs strictly less than

$$(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}) \cdot c_h$$

and

$$(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} p q_{\text{rank}[i]}) \cdot c_h$$

respectively (Theorems 4.5.8 and 4.8.5).

We concluded by showing a mistake in the Master's thesis of Thite, and subsequently proving our solutions provided an improvement over Thite's in the related HMM_2 model.

In Chapter 5, we considered the optimum BST problem without explicit ordering on the keys. This essentially left us with a multiset of probabilities over which we attempted to build an optimum BST. In section 5.2, we described an algorithm, GREEDY-MS and proved that it was within $\frac{n+1}{2n}$ of optimal under the Standard Model of cost for a multiset with n keys.

We have tabulated these results with novel contributions in bold.

Algorithm	Model	Running Time	Worst case expected cost
Modified Minimum Entropy	RAM	$O(n^2)$	$\mathbf{C \leq H + 4}$
ApproxMWPaging	HMM	$\mathbf{O(n)}$	$\mathbf{C < (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank[i]}}) \cdot c_h}$
ApproxBSTPaging	HMM	$\mathbf{O(n)}$	$\mathbf{C < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank[i]}}) \cdot c_h}$
GREEDY-MS	RAM	$\mathbf{O(n \cdot lg(n))}$	$\mathbf{OPTIMAL + 1}$

Table 6.1: Models, running rimes, and worst case expected costs for algorithms discussed in this thesis.

Several interesting and related problems still remain open. Firstly, is $H + 4$ a tight bound for the Modified Entropy Rule? We conjecture that this is not the case, and we believe the bound could be lowered to $H + 2$. Moreover, while the metric used to search for the root in this heuristic is good, it is definitely not perfect. The root chosen (up to the modifications of the rule) has the maximum 3-way entropy split. However, this is not necessarily the best root, as selecting larger probability keys as the root can decrease the cost of the tree. It would be interesting to consider what the correct metric would be for correctly selecting the root, possibly in a greedy manner.

The work in Chapter 4 can likely be extended to more recent models for external memory. We made attempts at extending these results to the Hierarchical Memory with Block Transfer Model of Aggarwal, Chandra, and Snir [3] which allows blocks of memory to be transferred from one location to another for a fixed cost. It would also be interesting to examine the problem under parallel memory models like those described in [56] or the Uniform Memory Hierarchy [5].

Chapter 5 introduces an interesting problem and a relatively simple greedy algorithm which solves the problem within $\frac{n+1}{2n}$ of optimal under the Standard Model. We have discussed several ideas for improvements to this GREEDY-MS algorithm. For example, when should a large probability be chosen as a leaf in our greedy top down algorithm?

Would this significantly effect our expected search cost? It would also be very interesting to find any polynomial time optimal solution under the Standard Model.

References

[1] M AdelsonVelskii and Evgenii Mikhailovich Landis. An algorithm for the organization of information. Technical report, DTIC Document, 1963.

[2] Alok Aggarwal, Bowen Alpern, Ashok Chandra, and Marc Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.

[3] Alok Aggarwal, Ashok K Chandra, and Marc Snir. Hierarchical memory with block transfer. In *28th Annual Symposium on Foundations of Computer Science*, pages 204–216. IEEE, 1987.

[4] Brian Allen and Ian Munro. Self-organizing binary search trees. *Journal of the ACM (JACM)*, 25(4):526–535, 1978.

[5] Bowen Alpern, Larry Carter, Ephraim Feig, and Ted Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2-3):72–109, 1994.

[6] ARM. *ARM7TDMI (Rev 3) Technical Reference Manual*, 2001.

[7] Paul Joseph Bayer. *Improved bounds on the costs of optimal and balanced binary search trees.* Massachusetts Institute of Technology, Project MAC, 1975. Master's Thesis.

[8] Rudolf Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4):290–306, 1972.

[9] Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141. ACM, 1970.

[10] Peter Becker. A new algorithm for the construction of optimal B-trees. *Algorithm Theory—SWAT'94*, pages 49–60, 1994.

[11] Peter Becker. Construction of nearly optimal multiway trees. In *Proceedings of the Third Annual International Conference on Computing and Combinatorics*, COCOON '97, pages 294–303, London, UK, UK, 1997. Springer-Verlag.

[12] Michael A Bender, Erik D Demaine, and Martin Farach-Colton. Cache-oblivious B-trees. In *41st Annual Symposium on Foundations of Computer Science*, pages 399–409. IEEE, 2000.

[13] Michael A Bender, Ziyang Duan, John Iacono, and Jing Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 29–38. Society for Industrial and Applied Mathematics, 2002.

[14] Andrew Donald Booth and Andrew JT Colin. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4):327–334, 1960.

[15] Prosenjit Bose and Karim Douïeb. Efficient construction of near-optimal binary and multiway search trees. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures*, pages 230–241. Springer-Verlag, 2009.

[16] Gerth Stølting Brodal and Rolf Fagerberg. Funnel heap—a cache oblivious priority queue. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, pages 219–228. Springer-Verlag, 2002.

[17] cppreference.com. std::map. http://en.cppreference.com/w/cpp/container/map. Accessed: 2016-02-02.

[18] Roberto De Prisco and Alfredo De Santis. On binary search trees. *Information Processing Letters*, 45(5):249–253, 1993.

[19] Erik D Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality—almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

[20] V Thomas Dock. *FORTRAN IV programming*. Reston, VA, 1972.

[21] E Knuth Donald. The art of computer programming. *Sorting and searching*, 3:426–458, 1999.

[22] Michael L Fredman. Two applications of a probabilistic search technique: Sorting x+ y and building balanced search trees. In *Proceedings of seventh annual ACM symposium on Theory of computing*, pages 240–244. ACM, 1975.

[23] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium Foundations of Computer Science.*, pages 285–297. IEEE, 1999.

[24] Robert G Gallager. *Information theory and reliable communication*, volume 2. Springer, 1968.

[25] Adriano M Garsia and Michelle L Wachs. A new algorithm for minimum cost binary trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.

[26] Edgar N Gilbert and Edward F Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

[27] Mordecai J Golin. personal communication, 2016-04.

[28] Mordecai J Golin, Claire Mathieu, and Neal E Young. Huffman coding with letter costs: A linear-time approximation scheme. *SIAM journal on computing*, 41(3):684–713, 2012.

[29] Leo Gotlieb. Optimal multi-way search trees. *SIAM Journal on Computing*, 10(3):422–433, 1981.

[30] David Graves and Chris Hogue. *Fortran 77 Language Reference Manual*. Silicon Graphics, Inc.

[31] Leo J Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science*, pages 8–21. IEEE, 1978.

[32] Reiner Güttler, Kurt Mehlhorn, and Wolfgang Schneider. Binary search trees: Average and worst case behavior. *Elektronische Informationsverarbeitung und Kybernetik*, 16:41–61, 1980.

[33] Jan Helbich. Direct selection of keywords for the kwic index. *Information Storage and Retrieval*, 5(3):123–128, 1969.

[34] Thomas N Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM (JACM)*, 9(1):13–28, 1962.

[35] Yasuichi Horibe. An improved bound for weight-balanced tree. *Information and Control*, 34(2):148–151, 1977.

[36] TC Hu. A new proof of the tc algorithm. *SIAM Journal on Applied Mathematics*, 25(1):83–94, 1973.

[37] T.C. Hu. *Combinatorial Algorithms*. Addison-Wesley, MA, 1982.

[38] TC Hu, Daniel J Kleitman, and Jeanne K Tamaki. Binary trees optimum under various criteria. *SIAM Journal on Applied Mathematics*, 37(2):246–256, 1979.

[39] Te C Hu and Alan C Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

[40] David A Huffman et al. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[41] Feffrey H Kingston. A new proof of the garsia-wachs algorithm. *Journal of Algorithms*, 9(1):129–136, 1988.

[42] DE Knuth. Sorting and searching.(the art of computer programming, vol. 3) addison-wesley. *Reading, MA*, pages 551–575, 1973.

[43] Donald E. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.

[44] James F Korsh. Greedy binary search trees are nearly optimal. *Information Processing Letters*, 13(1):16–19, 1981.

[45] James F Korsh. Growing nearly optimal binary search trees. *Information Processing Letters*, 14(3):139–143, 1982.

[46] Kenneth C Louden. Compiler construction. *Cengage Learning*, 1997.

[47] Kurt Mehlhorn. Sorting and searching, volume 1 of data structures and algorithms, 1984.

[48] Michael S Paterson and F Frances Yao. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, 13(1):99–113, 1992.

[49] Robert A Schumacker, Brigitta Brand, Maurice G Gilliland, and Werner H Sharp. Study for applying computer-generated images to visual simulation. Technical report, DTIC Document, 1969.

[50] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

[51] Haoyu Song, Murali Kodialam, Fang Hao, and TV Lakshman. Building scalable virtual routers with trie braiding. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[52] Shripad Thite. Optimum binary search trees on the hierarchical memory model. *arXiv preprint arXiv:0804.0940*, 2008.

[53] Vijay K. Vaishnavi, Hans-Peter Kriegel, and Derick Wood. Optimum multiway search trees. *Acta Informatica*, 14(2):119–133, 1980.

[54] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001.

[55] Jeffrey Scott Vitter and Mark H Nodine. Large-scale sorting in uniform memory hierarchies. *Journal of Parallel and Distributed Computing*, 17(1):107–114, 1993.

[56] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory, ii: Hierarchical multilevel memories. *Algorithmica*, 12(2-3):148–169, 1994.

[57] WA Walker and CC Gotlieb. *A Top Down Algorithm for Constructing Nearly-optimal Lexicographic Trees*. University of Toronto, Department of Computer Science, 1971.

[58] Peter F Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

[59] Raymond W Yeung. Alphabetic codes revisited. *Information Theory, IEEE Transactions on*, 37(3):564–572, 1991.